NIZKCTF: A Non-Interactive Zero-Knowledge Capture the Flag Platform

PAULO MATIAS, Federal University of São Carlos, Brazil
PEDRO BARBOSA, Federal University of Campina Grande, Brazil
THIAGO CARDOSO, Hekima, Brazil
DIEGO MARIANO, University of São Paulo, Brazil
DIEGO ARANHA, University of Campinas, Brazil

Capture the Flag (CTF) competitions are educational and professional tools for the cybersecurity community. Unfortunately, CTF platforms suffer from the same security issues as other software components, what may give advantage to competitors who target the actual platform instead of the challenges. While it is arguable that successful attacks against the platform demonstrate relevant skills, the organizers may be interested into enforcing rules and rewarding solutions of the contest problems, due to sponsorship duties or focused recruiting efforts. To mitigate this, we present NIZKCTF, the first open-audit CTF platform based on non-interactive zero-knowledge proofs. NIZKCTF is publicly available for anyone who wants to run a CTF competition and provides strong transparency guarantees through the protocol, allowing any entity to verify the contest progression and outcome by employing a Git-based transaction log, a continuous integration service and zero-knowledge proofs. Using NIZKCTF, we conducted a competition for 10 invited teams. This competition had a bug bounty program, with cash prizes for teams able to exploit and compromise the CTF result. In this experiment, we observed that attacks carried by the teams against the platform were unsuccessful.

ACM Reference Format:
Paulo Matias, Pedro Barbosa, Thiago Cardoso, Diego Mariano, and Diego Aranha. 2017. NIZKCTF: A Non-Interactive Zero-Knowledge Capture the Flag Platform. 1, 1, Article 1 (August 2017), 10 pages.
https://doi.org/10.1145/nnnnnn.nnnnnn

1 INTRODUCTION

A critical element of a robust cybersecurity strategy is having trained people in recent technological security issues. Research shows that the United States is the most prepared country against cyber attacks [1], however, there is also a problem of quantity and quality of professionals, especially when it comes to more sophisticated skills such as security by design, defensive programming, applied cryptography, threat intelligence and forensic analysis after a compromise [10].

In order to reduce the shortage of cybersecurity professionals, companies, schools, universities and military institutions have been promoting Capture the Flag (CTF) competitions around the world to foster the engagement of professionals in cybersecurity topics. CTF competitions are usually designed to serve an educational purpose to give participants experience with computer security problems from a wide spectrum of technical areas, as well as conducting
and reacting to the sort of attacks found in the real world. CTF competitions can also serve as an inexpensive recruiting tool to fill specific positions with highly-skilled talent. Reverse-engineering, exploitation, forensics, web, programming and cryptanalysis are among the required skills in CTF competitions.

There are two styles of CTF competitions: attack / defense and jeopardy. In an attack / defense competition, each team is given a machine to defend on an isolated network. Teams are scored based on both their success in defending their assigned machine and on their success in attacking the other team’s machines. Jeopardy competitions are more common and usually involve multiple categories of problems, each of which contains a variety of questions of different values and levels of difficulty. A correct solution to a problem reveals a flag, which is submitted to the scoring platform for points. Teams attempt to earn the most points in the competition’s time frame (e.g., 24 hours), but usually do not directly attack each other. Rather than a race, this style of game play encourages taking time to approach challenges and prioritizes quantity of correct submissions over the timing.

An important concern is to protect the main software platform against attacks. The platform is usually responsible for storing the flags and updating the scoreboard as the contest progresses. Unfortunately, because CTF platforms suffer from the same software security issues as any software component and due to incentives from the high competitiveness in such environments, it is common to find teams targeting the platform instead of the challenges. There are no independently verifiable guarantees that the teams really solved the challenges and the scoreboard is correct. Successful attacks against the platform arguably demonstrate relevant skills, but organizers may be more interested in enforcing the rules and rewarding solutions for the challenges, due to sponsorship duties or focused recruiting efforts.

In this paper, we propose a novel platform called NIZKCTF: Non-Interactive Zero-Knowledge Capture the Flag Platform. With NIZKCTF, there are no flags stored in the platform, and therefore, for a solved challenge, the team does not submit the flag itself, but a public zero-knowledge proof. This proof is specific for the team and the challenge. Since it is a public proof, the platform and other teams can audit and confirm that the team has indeed solved the challenge, without being able to deduce the flag. We implemented NIZKCTF as an open-source project and its architecture includes software elements such as a Git-based centralized repository (e.g., to commit the flags and receive the event updates), and a continuous integration system (e.g., to automatically merge all requests of teams in the repository of the competition). Any CTF promoter can instantiate and use NIZKCTF in their competition. To validate the usefulness and security of NIZKCTF, we conducted the Pwn2Win Platform Test Edition, a small CTF for 10 invited teams. This competition had a bug bounty program, with cash prizes for teams able to find vulnerabilities and compromise the CTF result. During and after the CTF, teams were not able to compromise the final result.

2 RELATED WORK

Concerns about securing a CTF platform against attacks or just making the competition more fair are not new. In 2015, the University of Birmingham and Imperial College London (UK) [7] announced a virtual machine (VM) containing vulnerable services and challenges: each student runs the VM locally and attempts to solve challenges inside the VM as they are made available. The solving of a challenge reveals a flag that is unique to a particular VM instance, allowing for the detection of collusion between students. As well as acquiring flags, students also had to provide traditional written answers to questions and sit an examination.

In 2015, the Carnegie Mellon University [5] released an automatic problem generator (APG) for CTF competitions, where a given challenge is not fixed, but rather can have many different automatically generated problem instances. APG offers players a unique experience and can facilitate deliberate practice where problems vary just enough to make...
sure a user can replicate the solution idea. APG also allows competition administrators to detect when users submit a copied flag from another user to the scoring server.

There are works regarding problems that may affect the overall quality of CTFs [6, 8, 9]. Chung et al. [8] present insights and lessons learned from organizing CSAW, one of the largest and most successful CTFs. Chapman et al. [6] present the competition design of PicoCTF, as well as an evaluation based on survey responses and website interaction statistics, and insights into the students who played. Despite the relevance, these works do not address solutions for the security problem of players attacking the CTF platform.

3 PROBLEM STATEMENT

Unfortunately, it is common to find vulnerabilities in CTF platforms. Here we present just two examples.

**RC3 CTF 2016**: For a moment during this CTF, the first place team had 3590 points. Suddenly, a team named "The board is vulnerable, please contact admin@seadog007.me" appeared in the scoreboard with 4500 points [19]. Figure 1 shows the record of this fact. Fortunately, the intention of the hackers who exploited the platform was just to report the vulnerability to the administrators. However, it was perfectly possible to exploit it for malicious purposes.

**CODEGATE CTF 2016 Finals**: During this CTF, a team discovered that the server hosting one of the challenges had an old kernel version and was vulnerable to *overlayfs* privilege escalation (CVE-2016-1576) [13]. The team members were able to gain root access and get some "free flags" (although they claim that they did not submit these flags until they really solved the challenges). Tracing the system calls of the SSH server, they waited for an administrator to log in and were able to get their password. Then, the team noticed that other servers (including the scoreboard) had the same password. After visiting the platform servers and having fun, they stopped the intrusion and proceeded to play as usual [12].

The aforementioned cases illustrate the impact of software vulnerabilities in CTF platforms and justify the ever growing importance of securing them. This paper addresses this problem and intends to answer the following questions: (i) how to guarantee integrity and a minimum level of fairness in a CTF competition, preventing teams from stealing flags by exploiting the platform? (ii) how to ensure auditability, allowing anyone to verify whether teams really solved the challenges according to the points presented in the scoreboard? Our proposed solution is discussed next.

4 COMPETITION PROTOCOL

This section describes the novel competition protocol employed by NIZKCTF. A threat model for CTF competitions is defined, followed by a formalization of the main theoretical tools and requirements for auditability.
4.1 Threat Model

In a CTF competition, the adversary is a player or team interested in exploiting vulnerabilities in the platform, instead of solving challenges, to obtain advantages such as stealing flags or manipulating the scoreboard. Due to the difficulty in developing a vulnerability-free system, NIZKCTF relies on cryptographic primitives to provide security properties, such as zero-knowledge proofs.

Another form of adversarial behavior is a team who wants to submit flags in place of another team without their consent (e.g., to harm a specific team by making it fall behind on the scoreboard). For this reason, NIZKCTF makes a zero-knowledge proof to be unique to a particular team.

The protocol alone does not protect against flag sharing, i.e., teams copying and submitting flags from others. For a solution that addresses this type of adversary, recall the automatic problem generation proposed by Burket et al. [5]. Since in a competition based on NIZKCTF it is possible to have automatic problem generation, our proposal can be extended to support protection against a flag sharing adversary.

4.2 Zero-knowledge Proof of Knowledge

Let \( L \) be a NP language such that \( pk_i \in L \) iff there exists a witness \( sk_i \) yielding \( M_L(pk_i, sk_i) = 1 \), where \( M_L \) is a polynomial-time Turing machine. Let us also assume that the probability of computing \( sk_i \) from \( pk_i \) in polynomial time is negligible.

A non-interactive zero-knowledge (NIZK) proof of knowledge [3] is a cryptographic scheme through which a prover knowing \( sk_i \) can convince a verifier of that fact, satisfying the following properties:

**Non-interactivity:** Access to a public set of common parameters and to the contents of the proof itself must be sufficient to verify a proof. Since no interaction with the prover is required, any party interested in acting as a verifier may do so.

**Completeness:** If \( pk_i \in L \) the proof generated by an honest prover knowing \( sk_i \) must be accepted by an honest verifier:

\[
\sigma = \text{Prove}(pk_i, sk_i) \implies \text{Verify}(pk_i, \sigma) = 1
\]

**Validity:** Any probabilistic polynomial-time (PPT) prover who does not know \( sk_i \) must have negligible probability of success \( \epsilon \) in convincing a verifier. Equivalently, for every possible PPT prover \( P \) (even malicious ones) there exists a knowledge extractor \( \text{Extract} \) that, given oracle access to \( P \), is able to extract \( sk_i \) with overwhelming probability \( 1 - \epsilon \) every time \( P \) succeeds in completing a new proof:

\[
\Pr[\sigma \leftarrow P(pk_i, sk_i); \; sk_i' \leftarrow \text{Extract}(pk_i, \sigma); \\
M_L(pk_i, sk_i') \vee ((pk_i, sk_i) \in Q) \vee \\
\neg \text{Verify}(pk_i, \sigma)] = 1 - \epsilon,
\]

where \( Q \) denotes a query tape which registers all previous queries that have been sent to a prover.

**Zero-knowledge:** The proof discloses no information about \( sk_i \) besides the fact that the prover knows its value. Equivalently, for every possible PPT verifier \( V \) (even malicious ones) there exists a simulator \( \text{Sim} \) that, given oracle access to \( V \), is able to convince \( V \) with a negligible difference in probability \( \epsilon \) when compared to an honest verifier.
prover, even though Sim has no knowledge of sk_i:

\[
\Pr[\sigma \leftarrow \text{Prove}(pk_i, sk_i) ; \ b \leftarrow \text{V}(pk_i, \sigma) : \ b = 1] - \Pr[\sigma \leftarrow \text{Sim}(pk_i) ; \ b \leftarrow \text{V}(pk_i, \sigma) : \ b = 1] \]
\[
= \epsilon.
\]

In NIZKCTF, values of pk_i are publicly disclosed, but the corresponding sk_i which allows proving pk_i \in L is kept secret. Every player holds a witness sk_t attesting membership to their team t. When a player solves a challenge c of the competition, they obtain a witness sk_c asserting they hold the answer to the challenge. In order to earn points for their team, the player needs to publicly prove simultaneous knowledge of sk_t and sk_c. The concept of simultaneous knowledge is formalized by performing proofs on an auxiliary NP language L' such that pk_t || pk_c \in L' iff there exists a witness sk_t || sk_c such that ML'(pk_t || pk_c, sk_t || sk_c) = ML(pk_t, sk_t) \land ML(pk_c, sk_c) = 1, where the operator || denotes string concatenation.

Different approaches exist for proving knowledge of witness sk_t || sk_c, but their practicality depends on the exact choice of ML and, consequently, of ML'. Next, we discuss such approaches.

4.2.1 A generic approach. The first approach consists in using a general-purpose non-interactive zero-knowledge (NIZK) proof system. If ML(pk_i, sk_i) is chosen to check whether H(sk_i) = pk_i, where H is a cryptographic hash function, then ML' could be implemented as an arithmetic circuit similar to the one adopted by the Zerocash protocol, so that proofs could be carried out using zk-SNARKs [16]. Generality, however, comes with a price: although zk-SNARKs are succinct and can be verified efficiently, players would still need to download a proving key of hundreds of MBs and consume tens of seconds of processing time to construct a proof.

4.2.2 A scheme based on digital signatures. The second approach, proposed and implemented in NIZKCTF, consists in choosing a ML(pk_i, sk_i) which verifies whether sk_i is the private key corresponding to the public key pk_i in a digital signature scheme. This choice allows us to reduce our proof of knowledge problem to that of digitally signing messages, whose implementation is simpler and more efficient than any known general-purpose NIZK proof system.

The Schnorr signature scheme and its key-prefixed variant over elliptic curves EdDSA [4] satisfy completeness, validity and zero-knowledge properties under the assumption that the discrete logarithm problem (DLP) is hard [11, 15]. However, incorrectly composing two signatures when constructing a proof of simultaneous knowledge may undermine the validity of such properties.

Let the following be the primitives of a secure digital signature scheme:

Sign(sk_i, m) = s || m

Signs the message m using the private key sk_i. Outputs the message m prepended to the signature s.

Open(pk_i, s || m) = \begin{cases} m, & \text{if } s \text{ is valid} \\ \bot, & \text{otherwise} \end{cases}

Verifies whether s is a valid signature for m produced by the private key sk_i corresponding to the public key pk_i. Outputs the original message m if the signature is valid, or \bot if it is invalid.
We propose the following scheme to prove knowledge of the witness \( sk_t \parallel sk_c \) corresponding to \( pk_t \parallel pk_c \in L' \):

\[
\text{Prove}(pk_t \parallel pk_c, sk_t \parallel sk_c) = \text{Sign}(sk_c, \text{Sign}(sk_t, c))
\]

(1)

\[
\text{Verify}(pk_t \parallel pk_c, \sigma) = \begin{cases} 
1, & \text{if } m = c \\
0, & \text{if } m \neq c,
\end{cases}
\]

(2)

where \( m = \text{Open}(pk_t, \text{Open}(pk_c, \sigma)) \).

We argue Eqs. 1 and 2 satisfy the properties of a NIZK proof of knowledge scheme:

**Non-interactivity:** Since \( pk_t, pk_c \) and the digital signature scheme parameters are public and known to all parties, the proof \( \sigma \) can be verified by Eq. 2 without interaction with the prover.

**Completeness:** Since \( \text{Open}(pk_i, \text{Sign}(sk_i, m)) = m \) for all \( i \) and for all \( m \) such that \( M_L(pk_i, sk_i) = 1 \), by simply substituting into Eqs. 1 and 2:

\[
\forall (t, c) \in L', \text{Verify}(pk_t \parallel pk_c, \text{Prove}(pk_t \parallel pk_c, sk_t \parallel sk_c)) = 1.
\]

**Validity:** If the digital signature scheme satisfies validity of the signed message \( s \parallel m \), there exists a knowledge extractor \( \text{Extract}(pk_i, s \parallel m) \) able to extract \( sk_i \) from the \( \text{Sign}(sk_i, m) \) operation implemented by any (possibly malicious) PPT prover \( P \). Therefore, a knowledge extractor \( \text{Extract}' \) able to extract \( sk_t \parallel sk_c \) from \( P \) is constructed as follows:

\[
\text{Extract}'(pk_t \parallel pk_c, s_t \parallel c) = \text{Extract}(pk_t, s_t \parallel c) \parallel \text{Extract}(pk_c, s_c \parallel c).
\]

Let \( Q \) be the query tape of \( \text{Sign}(sk_i, m) \), and \( Q' \) be the query tape of \( \text{Prove}(pk_t \parallel pk_c, sk_t \parallel sk_c) \). \( \text{Extract}' \) succeeds as long as:

- \((sk_t, c) \notin Q\):
- Otherwise, \( P \) may replay \( s_t \parallel c \) from a previous run, causing \( \text{Extract}(pk_t, s_t \parallel c) \) to fail in extracting \( sk_t \).
- \((sk_c, s_t \parallel c) \notin Q\):
- Otherwise, \( P \) may replay \( s_c \parallel s_t \parallel c \) from a previous run, causing \( \text{Extract}(pk_c, s_c \parallel s_t \parallel c) \) to fail in extracting \( sk_c \).

However, since all messages signed by \( sk_t \) reference \( c \), \((sk_t, c) \in Q \iff (pk_t \parallel pk_c, sk_t \parallel sk_c) \in Q' \). Similarly, since all messages signed by \( sk_c \) reference \( t \), \((sk_c, s_t \parallel c) \in Q \iff (pk_t \parallel pk_c, sk_t \parallel sk_c) \in Q' \).

The definition of validity allows the knowledge extractor to fail when \((pk_t \parallel pk_c, sk_t \parallel sk_c) \in Q' \), thus existence of \( \text{Extract}' \) proves that validity is satisfied.

**Zero-knowledge:** If the digital signature scheme satisfies zero-knowledge of the private key \( sk_i \), there exists a simulator \( \text{Sim}(pk_i, m) \) able to convince the validity of the signed message \( s \parallel m \) to the \( \text{Open}(pk_i, s \parallel m) \) operation implemented by any (possibly malicious) PPT verifier \( V \). Therefore, there exists a simulator \( \text{Sim}' \) able to convince \( V \) of the proof validity:

\[
\text{Sim}'(pk_t \parallel pk_c) = \text{Sim}(pk_c, \text{Sim}(pk_t, c)).
\]

4.2.3 **Requirements for the challenge witness**. Zero-knowledge of the witness \( sk_c \) is useful only as long as \( sk_c \) cannot be easily found by exhaustive search. Recall \( M_L \) and \( pk_c \) are public. Therefore, if \( sk_c \) does not have sufficient randomness, an offline brute-force attack has non-negligible chance of success in finding its value.
In some CTFs, the flag $f_c$ for a challenge $c$ consists of a random hexadecimal string large enough (e.g., 256 bits, after decoding) to make a brute-force attack unfeasible. In this case, the flag $f_c$ can be used directly as the seed for a deterministic digital signature key pair generator:

$$(sk_c, pk_c) = \text{KeyPair}(f_c)$$

Many competitions, however, adopt password-like flags, such as "CTF-{you_mastered_technique_Y}". In this case, a password-based key derivation function $\text{PBKDF}$ can be used along with a public salt value $\phi_c$ to increase the difficulty of an offline brute-force attack [18]:

$$(sk_c, pk_c) = \text{KeyPair}(\text{PBKDF}(\phi_c, f_c))$$ (3)

In NIZKCTF, we use Eq. 3 as a conservative choice to support both types of flags.

4.3 Auditability

In order to allow CTFs to be openly audited and independently verified, it is necessary to have all operations carried in a database and the following requirements met:

**History preservation:** The database must be able to recover a snapshot of its state after each committed transaction and preserve the logical order of these transactions.

**Immutability:** Once a transaction is committed, the database must prevent it from being erased. If an application needs to revert data to a previous state, the only way to perform that operation must be by performing a new transaction.

**Replication:** Anyone interested in auditing the competition must be able to retrieve and replicate the entire contents and transaction history of the database.

5 IMPLEMENTATION

Different instances of NIZKCTF can be constructed by choosing distinct underlying technologies. We selected components for implementing NIZKCTF with the goal of maximizing the usage of free-of-charge hosted services like GitHub (or GitLab) and Amazon cloud services which provide a permanent free tier (AWS Lambda and SNS).

Our implementation\(^1\) is composed by the following modules: a distributed storage for sharing data (implemented by a Git repository), a continuous integration script for accepting submissions (implemented by an AWS Lambda function), a command-line interface for interacting with the platform, and a web interface for displaying the list of challenges and the scoreboard.

As can be seen in Figure 2, the distributed storage is used for propagating the necessary data while keeping the full change history. Players then interact with the distributed storage by using the command-line interface, which implements the NIZKCTF protocol. A request to merge the new data with the main repository is created and later evaluated by the Lambda Function. The Lambda Function checks the validity of the modifications, then decides to accept or deny the request. After the changes are merged, the web interface starts using the most recent version of the data.

The distributed storage allows the replication of challenges, team registrations, proofs and other CTF metadata while ensuring that the entire change history is preserved. In our implementation, this property is achieved by adopting a

\(^1\)Available in https://github.com/pwn2wincftf/PTE
Fig. 2. Overview of our implementation. Players modify local Git repositories and create pull requests to the central repository. An AWS Lambda function is triggered to merge the pull request to the central repository if changes are valid.

Git repository as the database. Git commits are stored as a Directed Acyclic Graph that can be later queried [14]. This allows any participant to audit all changes made, including ordering and timestamps.

When changes are made to the player’s local storage, a pull request is created to merge the modifications with the central Git repository managed by the competition’s staff. The pull request is evaluated by the Lambda Function and, if accepted, all changed data is incorporated and can be propagated to other teams.

In order to avoid tampering with commit history, the repository is configured to disallow force pushes. Without force pushes, changes committed to the central repository must always descent from the central repository history. In other words, pushed commits are not allowed to modify the commit chain, guaranteeing immutability of previously committed changes.

The Lambda Function works like a continuous integration service. It is triggered by a GitHub hook to automatically accept pull requests containing team registrations and proof submissions. However, since the Lambda Function does not have access to any privileged information about the challenges, any node with access to the distributed storage can also be used for verifying submissions.

The command-line interface is a Python script used for automating modifications on the distributed storage and uses libsodium for all cryptography implementations. Currently, the following operations are supported:

**Login:** Connects to GitHub or GitLab, generates an API token and creates a fork of the CTF’s main repository.

**Register:** Registers a new team and creates a pull request.

**Challenges:** Lists available challenges with their title, description, categories and rewards.

**Submit:** Checks if the challenge’s private key can be successfully computed from the flag provided by the competitor, then generates a submission request of the zero-knowledge proof.

**Score:** Reads the accepted submissions file and presents the scoreboard.
The web interface uses the repository as a GitHub page which exposes files with an HTTP server. This allows the challenges and scoreboard to be viewed in a more user-friendly way. The interface is implemented using only client side programming languages (HTML, CSS and Javascript). Challenges and scoreboard files are loaded using Ajax in order to give a dynamic feel.

It is worth noting that the distributed storage could also be implemented using a blockchain like Ethereum [17]. This would allow a fully distributed implementation of NIZKCTF in which submission proofs are appended to the blockchain and validated by contest participants. In this paper this was not the chosen implementation because it would require a small transaction fee for each submitted proof.

6 VALIDATION

To validate our proposal, we conducted the Pwn2Win Platform Test Edition, a small CTF competition for 10 invited teams. The objective of this CTF was to assess the usefulness and security of NIZKCTF. In order to achieve that, we used the Goal, Question, Metric (GQM) paradigm [2], a mechanism for defining and evaluating goals using measurement.

GQM defines a measurement model on three levels: conceptual level (Goal), operational level (Question) and quantitative level (Metric). GQM templates are a structured way of specifying goals and contains the following fields: purpose, object of study, focus, stakeholder and context. Here is a GQM template to express the goal of our study: The purpose of this study is to **evaluate the usefulness of NIZKCTF when being used by the participants in a CTF competition**.

To characterize the measurement object, we defined the research question **RQ1**: Is our implementation of NIZKCTF able to provide the features (e.g., challenges, submissions and scoreboard) of a common CTF?

Pwn2Win Platform Test Edition had 7 challenges and duration of 12 hours. There were one challenge of exploitation, cryptography, web, networking and miscellaneous, and two of reverse engineering. Teams were able to solve the challenges and there was a scoreboard, just like a common CTF. Players also gave good feedbacks and no one had objections in using NIZKCTF in future CTFs. Therefore, we support a positive answer for **RQ1**.

From the 10 invited teams, 5 of them scored (solved at least one challenge). Since the CTF had many low-complexity challenges and the invited teams were very experienced (for example, one of the teams was the 2016 second place at ctftime.org), we assume that teams that did not score were focused in trying to exploit the platform, as we present next.

Here is another GQM template to express another goal of our study: The purpose of this study is to **evaluate the security of NIZKCTF against attacks to the platform from the participants in a CTF competition**.

To characterize the measurement object, we defined the research question **RQ2**: Is any participant able to attack the platform and compromise the CTF result?

To answer this question, we made a bug bounty program, with 450 BRL in cash prizes for teams who find vulnerabilities and compromise the result. During and after (we kept the platform online for 20 days) the CTF, teams were not able to do that. Therefore, we support a negative answer for **RQ2**.

7 CONCLUSIONS

With the growing interest in CTFs, there is a need for a secure and auditable platform. We presented a novel platform called NIZKCTF: Non-Interactive Zero-Knowledge Capture the Flag. Through theoretical cryptography constraints, we claim that a CTF running NIZKCTF is more secure than when running a traditional platform.

An implementation of NIZKCTF was tested in the Pwn2Win Platform Test Edition, and we also claim that it was the first CTF to use a zero-knowledge protocol for proving the challenges’ resolution. It was a competition with a bug...
bounty program and cash prizes. During and after the competition, teams were not able to exploit the platform and, therefore, they did not receive any prize from the bug bounty program.

As a future work, we intend to continue using our proposal in upcoming competitions. We also intend to experiment with different instances of NIZKCTF, e.g., replacing the central storage by a fully peer-to-peer mechanism.

REFERENCES

[1] ABI Research. 2015. Global Cybersecurity Index & Cyberwellness Profiles. Technical Report. International Telecommunications Union, Geneva, CH.

[2] V. R. Basili. 1992. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report. University of Maryland at College Park, College Park, MD, USA.

[3] M. Bellare and O. Goldreich. 1993. On Defining Proofs of Knowledge. In 12th Annual International Cryptology Conference. Springer Berlin Heidelberg, Santa Barbara, CA, USA, 390–420. https://doi.org/10.1007/3-540-48071-4_28

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang. 2012. High-speed high-security signatures. Journal of Cryptographic Engineering 2, 2 (2012), 77–89. https://doi.org/10.1007/s13389-012-0027-1

[5] J. Burlet, P. Chapman, T. Becker, C. Ganas, and D. Brumley. 2015. Automatic Problem Generation for Capture-the-Flag Competitions. In 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (SGSE 15). USENIX Association, Washington, DC, USA.

[6] P. Chapman, J. Burlet, and D. Brumley. 2014. PicoCTF: A Game-Based Computer Security Competition for High School Students. In 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (SGSE 14) USENIX Association, San Diego, CA, USA

[7] T. Chothia and C. Novakovic. 2015. An Offline Capture The Flag-Style Virtual Machine and an Assessment of Its Value for Cybersecurity Education. In 2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (SGSE 15) USENIX Association, Washington, DC, USA. https://www.usenix.org/conference/sgse15/summit-program/presentation/chothia

[8] K. Chung and J. Cohen. 2014. Learning Obstacles in the Capture The Flag Model. In 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (SGSE 14) USENIX Association, San Diego, CA, USA. https://www.usenix.org/conference/sgse14/summit-program/presentation/chung

[9] A. Davis, T. Leek, M. Zhivich, K. Gwinnup, and W. Leonard. 2014. The Fun and Future of CTF. In 2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (SGSE 14) USENIX Association, San Diego, CA, USA. https://www.usenix.org/conference/sgse14/summit-program/presentation/davis

[10] K. Evans and F. Reeder. 2010. A Human Capital Crisis in Cybersecurity: Technical Proficiency Matters. Technical Report. Center for Strategic and International Studies, Washington, DC, USA.

[11] E. Kiltz, D. Masny, and J. Pan. 2016. Optimal Security Proofs for Signatures from Identification Schemes. In 36th Annual International Cryptology Conference. Springer Berlin Heidelberg, Santa Barbara, CA, USA, 33–61. https://doi.org/10.1007/978-3-662-53008-5_2

[12] More Smoked Leet Chicken. 2016. CODEGATEgate. http://mslc.ctf.su/wp/codegategate. (May 2016).

[13] NIST. 2016. Computer Security Resource Center - National Vulnerability Database. https://nvd.nist.gov/vuln/detail/CVE-2016-1576. (Feb. 2016).

[14] B. O’ Sullivan. 2009. Making Sense of Revision-Control Systems. Commun. ACM 52, 9 (Sept. 2009), 56–62. https://doi.org/10.1145/1562164.1562183

[15] D. Pointcheval and J. Stern. 2000. Security Arguments for Digital Signatures and Blind Signatures. Journal of Cryptology 13, 3 (2000), 361–396. https://doi.org/10.1007/s001450010003

[16] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy. San Jose, CA, USA, 459–474. https://doi.org/10.1109/SP.2014.36

[17] G. Wood. 2017. Ethereum: A secure decentralised generalised transaction ledger. https://ethereum.github.io/yellowpaper/paper.pdf. (Jan. 2017).

[18] F. F. Yao and Y. Lisa. Yin. 2005. Design and Analysis of Password-Based Key Derivation Functions. In RSA Conference 2005. Springer Berlin Heidelberg, San Francisco, CA, USA, 245–261. https://doi.org/10.1007/978-3-540-30574-3_17

[19] J. Yu. 2016. Seadog GitHub repository: RC3-CTF-2016-scoreboard. https://github.com/seadog007/RC3-CTF-2016-scoreboard. (Nov. 2016).