یازدهمین کنگره ملی سراسری
فناوریهای نوین در حوزه توسعه پایدار ایران
11th National Congress of
the New Technologies in Sustainable Development of Iran
senaconf.ir

رویکردی نوین برای جلوگیری از آسیب پذیری در قراردادهای هوشمند و مقابله با حملات Reentrancy بر بستر بلاک چین

محمودرضا پارسائیان[۱]، حسین صمیمی[۲]

[۱] دانشگاه تهران – پردیس بین الملل کیش، تهران mahmoud.parsaeian@ut.ac.ir

[۲] پژوهشگاه ارتباطات و فناوری اطلاعات،hossein-samimi@itrc.ac.ir

چکیده

بیش از یک دهه از معرفی فناوری بلاک چین و پروتکل بیت کوین توسط ساتوشی ناکاموتو در سال ۲۰۰۸ می گذرد. پروتکل بیت کوین به عنوان یک نسخه ضعیف از مفهوم قرارداد هوشمند شناخته می شود و امکان برنامه نویسی در آن وجود ندارد. در راستای رفع این محدودیت، ارز دیجیتالی اتریوم (Ethereum) توسط ویتالیک بوترین و گاوین وود در سال ۲۰۱۴ معرفی شد. اتریوم اجازه می‌دهد تا قراردادهای هوشمند پیچیده بین طرف‌های غیرقابل اعتماد، ایجاد و اجرا شوند. قراردادهای هوشمند با طیف وسیعی از شرکت ها مانند خدمات مالی، دفتر اسناد رسمی، املاک و اینترنت اشیا و مانند آنها سازگار هستند. از آنجا که ساختار بلاک چین بصورت توزیع شده است و حجم بالائی از پول در این معاملات جابجا می شود. در نتیجه برای بهره مندی از قابلیت های قراردادهای هوشمند چالش های بسیاری مانند احراز هویت ،امنیت تراکنش، محرمانه بودن، مسائل مربوط به حریم خصوصی و همچنین مقابله با حملات امنیتی مختلف وجود دارد که رفع آنها نیازمند تحقیقات بیشتری درآینده است.

هدف از این مقاله، معرفی پرخطر ترین حمله سایبری بر روی قراردادهای هوشمند بر بستر بلاک چین با عنوان حمله ورود مجدد (Reentrancy) و ارایه راهکاری کارآمد برای جلوگیری از آن می باشد. در این چارچوب ابتدا یک نسخه از بلاک چین بر بستر EVM (Ethereum Virtual Machine) بنام BankBalance با قرارداد هوشمندی که با کد هائی به زبان برنامه نویسی شی گرای سطح بالای Solidity ، نوشته شده است، راه اندازی کرده ایم. این نسخه امکان کم کردن (Withdrew) یا اضافه کردن (Deposit) روی فیلد موجودی (balance) را در محیط وب در تعامل با بلاک چین، فراهم می کند. در قدم بعد نشان داده ایم که این بلاک چین در مقابل حمله ورود مجدد آسیب پذیر است و سپس با تحلیل این آسیب پذیری و شناسایی دلایل بروز آن، کدهای تکمیلی توسعه یافته و اقدامات اصلاحی لازم بر روی قرارداد هوشمند موجود انجام داده ایم. نتایج حاصل از بررسی های انجام شده نشان می دهد که آسیب پذیری موجود برطرف شده و بلاک چین جدید توسعه یافت از حمله ورود مجدد در امان است.

واژه‌های کلیدی

Blockchain, Smart contracts, Vulnerabilities of Smart Contract, Distributed ledger, Attacks on Smart Contracts

۱. مقدمه

امروزه بلاک چین به عنوان سکویی برای تحول دیجیتالی در تمام کسب و کارها مطرح می باشد. با توجه به خاصیت توزیع شدگی و استفاده موثر از توابع چکیده ساز در کنار تکنیک های رمزنگاری، بلاک چین سابقه تراکنش های اطلاعاتی هر دارایی دیجیتال را نگه می دارد و آنرا غیرقابل تغییر و شفاف می سازد. بلاک چین تکه های داده رمزگذاری شده را قبل از اینکه آنها به هم متصل شوند، ذخیره می کند. در بلاک چین اطلاعات دارائی های دیجیتال به جای تکرار یا جابجایی یا تغییر، بصورت توزیع شده منتشر می شوند و رکوردی تغییرناپذیر از دارایی ایجاد می کنند. دارایی غیرمتمرکز است و دسترسی عمومی را در هر زمان واقعی و شفاف فراهم می سازد. امنیت و یکپارچگی اطلاعات (Integrity) از طریق ثبت کردن شفاف تغییرات در زمان هر تغییر حفظ می شود [۱]. افزایش امنیت که به لطف داشتن دفاترکل عمومی بدست می آید، این تکنولوژی را به یک فناوری ضروری در هر صنعتی مانند خدمات مالی، دفتر اسناد رسمی، املاک و مستغلات و اینترنت اشیا تبدیل کرده است.

2.1. مفاهیم کلیدی در بلاک چین

سه مفهوم کلیدی در بلاک چین وجود دارد:

الف). بلوک ها Blocks : شامل ، بلوک داده، هش با ۲۵۶ بیت Hash ، و نانس با ۳۲ بیت Nonce

هر تراکنش به عنوان یک "بلوک" از داده ها به محض وقوع ثبت می شود. این تراکنش ها حرکت یک دارایی مشهود یا نا مشهود را به تصویر می کشد. از بلوک داده می توان برای ذخیره هر گونه اطلاعات استفاده کرد. هر بلوک به بلوک‌هایی که قبل از آن و آنهایی که بعد از آن آمده‌اند پیوند داده می‌شود. با انتقال دارایی از مکانی به مکان دیگر یا تغییر مالکیت، بلوک ها ثبت می شوند و این بلوک ها یک زنجیره داده را تشکیل می دهند. بلوک‌ها زمان و ترتیب دقیق تراکنش‌ها را تأیید می‌کنند و به طور ایمن به یکدیگر متصل می‌شوند تا از تغییر یا درج هر بلوکی بین دو بلوک دیگر جلوگیری شود. تراکنش های روی بلاک چین در یک زنجیره برگشت ناپذیر به هم مرتبط می شوند. هر بلوک متوالی تأییدیه بلوک قبلی است و در نتیجه کل زنجیره بلوک را تأیید می کند، و در نتیجه دستکاری در بلاک چین آشکار می شود. این قدرت کلیدی یعنی تغییرناپذیری از مزایای بلاک چین است. تغییر ناپذیری خطر دستکاری توسط یک بازیگر بد را از بین می برد و با ایجاد یک دفتر کل تراکنش های قابل اعتماد برای کاربران شبکه ایجاد می کند [۲]. هش تابعی است که نیازهای رمزگذاری شده مورد استفاده برای محاسبات بلاک چین را برآورده می کند. از آنجایی که ، حدس زدن طول هش عملا غیر ممکن و سخت است، کسی نمیتواند بلاک چین را شکست دهد. هش ها طول ثابتی دارند. و مقدار هش شده همیشه برای همان داده ها یکسان خواهد بود. از اطلاعات موجود در هدر بلوک برای تولید هش [۱] استفاده می شود. نانس nonce در بلاک چین عددی است که به یک بلوک هش شده (رمزگذاری شده) در یک بلاک چین اضافه می شود [۱].

ب). ماینرها: ماینرها از تکنیکی به نام ماینینگ و محاسبات ریاضی برای اضافه کردن بلوک های جدید به زنجیره بلاک چین استفاده می کنند.

ج). گره ها: هر نوع دستگاه تکنولوژیکی که بتواند تراکنش های کپی شده بلاک چین را ردیابی کند [۲]، [۳].

اتریوم (Etherum) یک پلتفرم بلاک چین است که دارای ارز دیجیتال مخصوص به خود بنام اتر (ETH) است و همچنین زبان برنامه نویسی خاص خود بنام، Solidity است. اتریوم یک پلتفرم منبع باز برای توسعه و به اشتراک گذاری برنامه های کاربردی شرکتی، مالی و سرگرمی است.

برای استفاده از dApps (برنامه های غیرمتمرکز و توزیع شده)، کاربران اتریوم باید برای هر تراکنش هزینه ای بپردازند. در اتریم واحد هزینه ها به عنوان "گاز" (Gas) نامیده می شود و بر اساس میزان قدرت پردازش استفاده شده، متفاوت است. در اصل اتر یا ETH یک ارز دیجیتال مرتبط با اتریوم است [۳] . ارز دیجیتال اتریوم در حال حاضر از نظر ارزش بازار پس از بیت کوین در رتبه دوم قرار دارد [۴].

مکانیسم اجماع (Consensus Mechanism) پروتکلی است که همه همتایان (Network Peers) شبکه را قادر می سازد تا در مورد وضعیت فعلی یک دفتر کل توزیع شده (DLT) به توافق برسند. در اصل، یک مکانیسم اجماع تضمین می‌کند که هر بلوک جدیدی که به یک بلاک چین اضافه می‌شود تا زنجیره بلاک ها تکمیل شود ، تنها نسخه حقیقتی است که همه گره‌های زنجیره روی آن توافق دارند.

رایج ترین اشکال الگوریتم های اجماع به شرح زیر است:

اثبات کار (PoW): اتریوم، مانند بیت کوین، در حال حاضر از یک پروتکل اجماع به نام اثبات کار (PoW) استفاده می کند. این امر به گره های شبکه اتریوم اجازه می دهد تا در مورد وضعیت تمام اطلاعات ثبت شده در بلاک چین اتریوم به توافق برسند. این پروتکل از انواع خاصی از حملات مالی جلوگیری می کند [۵].

اثبات سهام (PoS): در اتریوم مکانیزم زیربنایی وجود دارد که اعتباردهنده ها (تائید کنندکان) را پس از دریافت سهام کافی فعال می کند. کاربران برای تبدیل شدن به یک تائید کننده باید ۳۲ سهم ETH ، سهام داشته باشند [۶].

اثبات سهام واگذار شده (DPoS): یک نوع محبوب از ایده اثبات سهام (PoS) است که در آن کاربران شبکه رای می‌دهند و نمایندگانی را برای اعتبارسنجی و تائید بلوک بعدی انتخاب می‌کنند. و به جای انتقال مستقیم توکن‌های خود به کیف پول دیگری، آنها را از طریق یک ارائه‌دهنده خدمات سهام انجام می‌دهند [۷].

2. قراردادهای هوشمند (Smart Contracts)

قراردادهای هوشمند از فناوری بلاک چین برای اعتبارسنجی، تأیید، اخذ کردن (گرفتن) و اجرای توافقات چند طرفه استفاده می‌کنند و امکان ساخت بلاک چین جدید را فراهم می‌آورند [۸]. قراردادهای هوشمند در بلاک چین به طرف های ناشناس اجازه می دهد تا بدون نیاز به یک مقام تصمیم گیرنده مرکزی، یا تحت تاثیر قرار گرفتن توسط فشار خارجی و یا سیستم قضایی، تراکنش ها و توافقات انجام پذیرد. به این ترتیب تراکنش ها شفاف، برگشت ناپذیر و قابل ردیابی هستند [۸]. در یک جمله، یک قرارداد هوشمند یک سیستم خود اجرا و خود اجبار است که توسط شرایط و ضوابط روشن تنظیم می شود و امکان ذخیره و اجرای تعهدات قراردادی از طریق بلاک چین را فراهم می کند.

از آنجایی که تمام داده های ثبت شده در بلاک چین غیرقابل تغییر و ایمن هستند، این موضوع یک تنظیم ایده آل برای قراردادهای هوشمند است. اطلاعات ذخیره شده در بلوک های یک قرارداد هوشمند رمزگذاری شده و در یک دفتر کل موجود است، به این معنی که هرگز نمی توان آن را از دست داد، اصلاح کرد یا پاک کرد [۸]. بهترین زبان های برنامه نویسی برای نوشتن قراردادهای هوشمند، زبان های برنامه نویسی شی گرا سطح بالا مانند Solidity، C++، JavaScript، Java و Golang هستند. [۹].

3. تحقیقات قبلی

کارهای تحقیقاتی مختلفی در رابطه با قراردادهای هوشمند و بلاک چین و روش های جلوگیری از حملات بر قراردادهای هوشمند انجام شده است. در [10] نویسندگان چارچوب، مکانیسم‌های عملیاتی، پلتفرم‌ها و زبان‌های برنامه‌نویسی را معرفی کرده اند و حملات ورورد مجدد و تراکنش های (TOD) را مورد بررسی و تجزیه و تحلیل قرار داده اند. در [11] نیاز به رشته مهندسی نرم افزار بلاک چین، مورد تاکید قرار گرفته که به مسائل ناشی از برنامه نویسی قراردادهای هوشمند و سایر برنامه های کاربردی در حال اجرا بر روی بلاک چین می پردازد. در [12] روشی بنام قانون "حفاظت" ، که مبتنی بر یک تحلیلگر منطق فازی است، معرفی شده است، نویسندگان این کار را برای شناسایی خودکار رایج ترین اشکالات ورود مجدد در قراردادهای هوشمند اتریوم ارائه کرده اند. در [13] نویسندگان حملاتی را به گره‌های اتریوم معرفی کردند که این حملات با استفاده از شبکه همتا به همتا برای سوء استفاده از همسایگان همتا (گره های دارای اعتبار یکسان) مورد استفاده قرار می گیرد. در [14] بر روی آسیب پذیری های امنیتی سمت Client یعنی نسخه (Go-Ethereum) یا (geth) تمرکز شده و به معرفی آسیب‌پذیری‌هائی که به دلیل طراحی نا امن API (واسطهای برنامه کاربردی) و مکانیزم خاص کیف پول اتریوم رخ می دهد، پرداخته شده است. مشکل ورود مجدد در [15] با معرفی روشی به نام Sereum مورد بررسی قرار گرفته است. در

[16]یک تکنیک ردیابی مبتنی بر تجزیه و تحلیل داده ها برای تجزیه و تحلیل حمله Overflow به نام EASYFLOW معرفی شده که می تواند از قراردادهای هوشمند در برابر حملات سرریز overflow محافظت کند. در [17] نیز با معرفی تکنیک بهره برداری (Exploitation) ، نویسندگان تکنیک های بهره برداری از بلاک چین را بر اساس منطق حمله به ۴ دسته *حمله به پروتکل های اجماع، اشکالات در قرارداد هوشمند، بدافزارهای در حال اجرا در سیستم عامل و کاربران متقلب* طبقه بندی کرده و هفت تکنیک مهم حمله را بررسی نموده اند.

دو کار مهم و زیربنایی نیز در سالهای ۲۰۲۰ و ۲۰۲۱ انجام شده است. در [18] نویسندگان طبقه بندی از آسیب پذیری هایی قرارداد هوشمند را با ارائه کد های برنامه نویسی فهرست کرده اند و به بررسی نحوه سوء استفاده مهاجمان از قراردادهای هوشمند پرداخته اند . در [19] نیز نویسندگان بر آسیب پذیری های قراردادهای هوشمند متمرکز شده اند و علت اصلی حملاتی مانند حمله بازگشت مجدد، حمله آدرس کوتاه/پارامتر، حمله انکار سرویس DoS، حمله اطلاعات از دست رفته درحین انتقال، محدودیت اندازه پشته/عمق پشته و حمله ورود مجدد در بلاک چین اتریم را مشخص کرده اند.

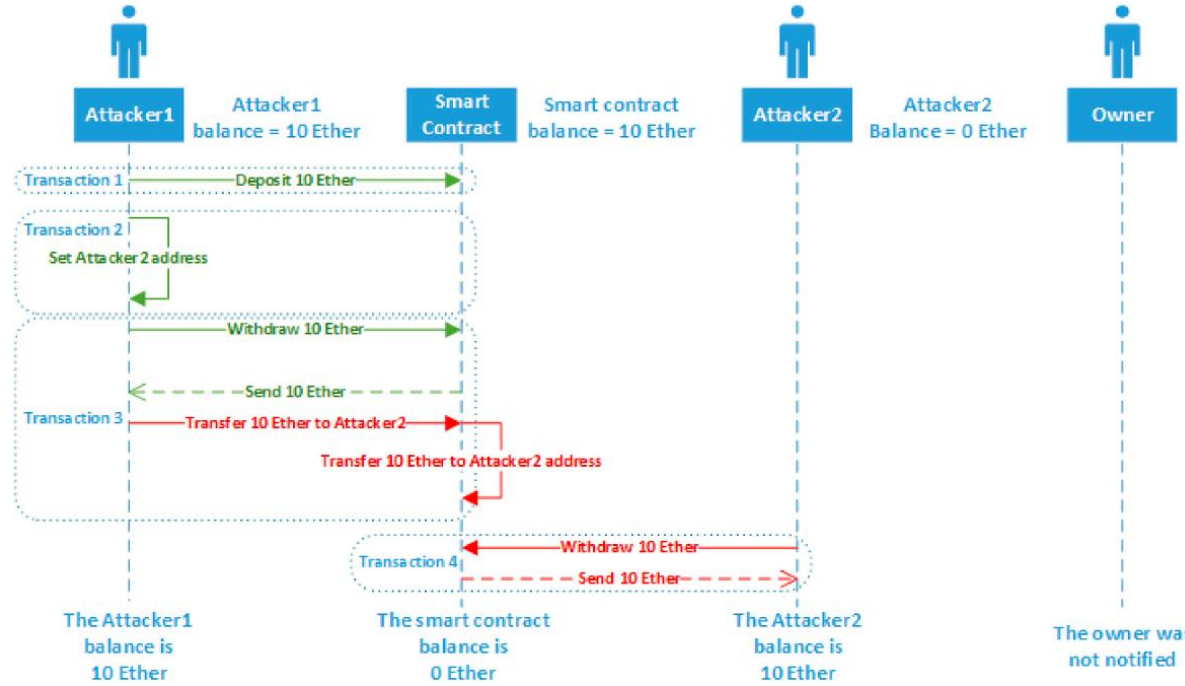
همه مطالعات قبلی به مشکلات مربوط به استفاده گسترده از فناوری بلاک چین می پردازند، اما هیچ یک از آنها به طور خاص به چگونگی استفاده از روشی برای بهبود راه کار های قرارداد هوشمند نمی پردازند. همانطور که قبلاً ذکر شد امنیت در قراردادهای هوشمند به دلیل ماهیت آنها بسیار مهم است. تجزیه و تحلیل صحیح و دقیق یک سیستم که می خواهیم بر روی بلاک چین اجرا کنیم ، در کنار برنامه نویسی بدون نقص و اشکال برای انجام آن از اهمیت استراتژیک برخوردار است. در ادامه ضمن معرفی مشکل، به ارائه راهکاری جدید برای مقابله با آن پرداخته شده است.

۴. معرفی مشکل و راهکاری برای جلوگیری از وقوع آن

همانطور که در پاراگراف های گذشته بیان شد ، با توجه به این نکات که هر وقت یک قرارداد هوشمند کامپایل می شود ، بصورت کد بر روی بلاک چین قرار می گیرد و همچنین قراردادهای هوشمند خود اجرا و خود اجبار هستند یعنی به محض اینکه شرایط قرارداد مهیا شد اجرا می شوند. و با در نظر داشتن وجود خاصیت های تغییر ناپذیری و فقط اضافه شدن رکورد یا بلوک در بلاک چین ، میزان اهمیت قراردادهای هوشمند مشخص می شود. بدین معنی که اگر در قرار داد هوشمند ما، اشکالی وجود داشته باشد، هکر ها از این نقص بیشترین بهره برداری را انجام خواهند داد.

۱.۴. حمله ورود مجدد یا Reentrancy چیست؟

حمله Reentrancy یکی از مخرب ترین حملات در قرارداد های هوشمند Solidity است. حمله ورود مجدد زمانی اتفاق می‌افتد که یک تابع در حین اجرا، یک قرارداد دیگری را فرا می خواند، (بخاطر ماهیت قراردادهای هوشمند **ناشناس است**)، سپس قرارداد غیرقابل اعتماد (ناشناس) در تلاش برای برداشتن وجوه موجود یا تعییر داد ه ها، یک فراخوان بازگشتی به تابع اصلی ایجاد می کند. هنگامی که قرارداد ما نتواند وضعیت خود را قبل از ارسال داده ها به روز کند، مهاجم می تواند به طور مداوم تابع برداشت را برای برداشتن وجوه (یا تعییر داده ها) در این قرارداد فراخوانی کند و مقادیر داد ه ها را به دلخواه خود تغییر دهد (شکل ۱). یک حمله Reentrancy مشهور در دنیای واقعی، حمله DAO است که باعث ضرر ۶۰ میلیون دلاری در آمریکا شد [۱۲] .



شکل ۱. چگونگی حمله ورود مجدد (Akhalifah et al., 2021)

۲,۴. چگونه از قراردادهای هوشمند در برابر حملات ورود مجدد محافظت کنیم؟

برای جلوگیری از حمله ورود مجدد در قرارداد هوشمند بزبان Solidity باید:

۱) اطمینان حاصل کنیم که همه State Variables ها قبل از فراخوانی قراردادهای خارجی، مقدار دهی و بروزرسانی می شوند.

۲) از اصلاح کننده های عملکردی Function Modifier استفاده کنیم که از ورود مجدد یا فراخوانی مجدد جلوگیری می کنند. Function Modifier ها ، در اصل کدهای برنامه هستند که می‌توانند قبل و/یا بعد از فراخوانی تابع اجرا شوند.

ما با تمرکز بر ساختار قراردادهای هوشمند در زبان برنامه نویسی Solidity و ایجاد کد های برنامه نویسی ، نشان دادیم که می توان از حملات ورود مجدد یا Reentrancy جلوگیری کرد.

۵. معرفی زیر ساخت ها و چگونگی کاربرد آنها

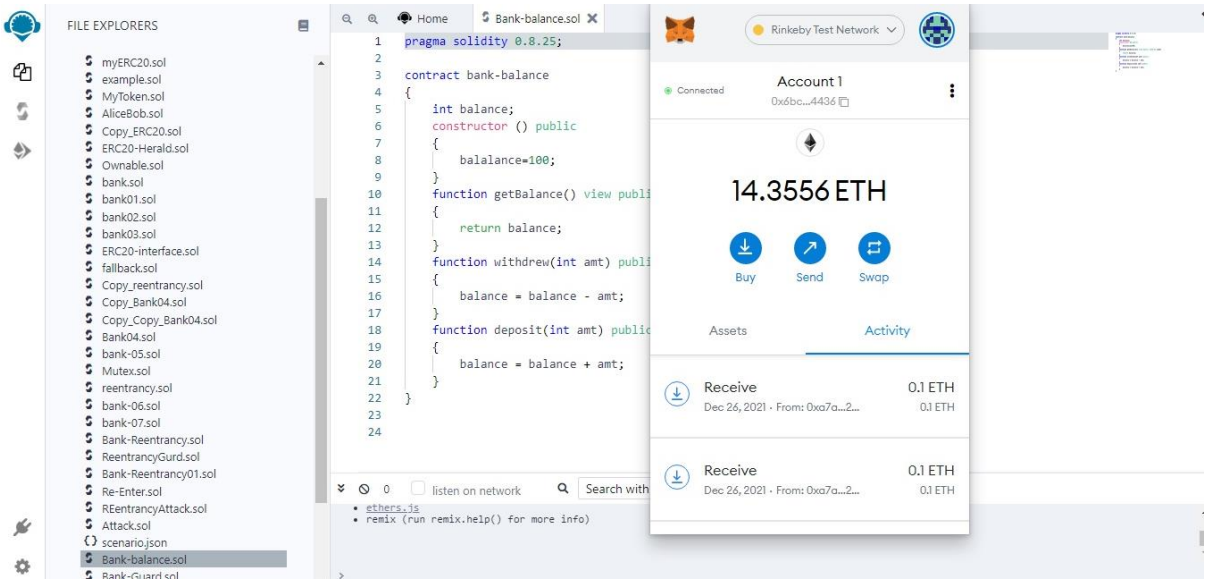
برای ایجاد و استفاده از یک بلاک چین برای نمونه های آزمایشی و دانشگاهی پلتفرم های مختلفی وجود دارد. ما از Remix IDE که یک برنامه کاربردی و تحت وب و منبع باز برای Etherum است ، استفاده کرده ایم. در Remix IDE می توانیم برنامه قرارداد هوشمند را که بزبان Solidity نوشته ایم را Compile و Deploy کنیم [20]. ما برای بوجود آوردن محیط Client/Server برای نوشتن و اجرای برنامه ها و سپس تعامل با بلاک چین Ethereum به نرم افزارهای Visual Studio Code برای نوشتن کد و بوجود آوردن Local Host نیاز داریم [۲۱] ، همینطور به NPM که یک بسته مدیریت همزمان در زمان اجری برنامه است (Runtime) در محیط جاوا اسکریپت که برای ارتباط با Node.js می باشد ، نیازمندیم [22] ، Node.js یک محیط اجراییِ سمت سرورِ متن‌باز و چندپلتفرمی برای زبان جاوا اسکریپت است [23]. و همچنین به web3.js [24] که مجموعه ای از کتابخانه ها ست که به ما امکان می دهد با استفاده از محیط HTTP، IPC یا WebSocket با یک گره (node) اتریوم محلی یا راه دور تعامل داشته باشیم ، نیاز داریم.

با توجه به اینکه برای انجام هر تراکنش در بلاک چین نیاز به پرداخت هزینه برای آن تراکنش هستیم ، ما به یک کیف پول مجازی مانند Metamax نیاز داریم. پس برای خود یک کیف پول ایجاد کرده و آنرا با رمز ارز اتریم ETH شارژ می کنیم [25]. کیف پول، Metamax به ما جهت امنیت نقل و انتقال دارائی ها ، یک زوج کلید عمومی و خصوصی می دهد که از این زوج کلید برای انجام تعاملات

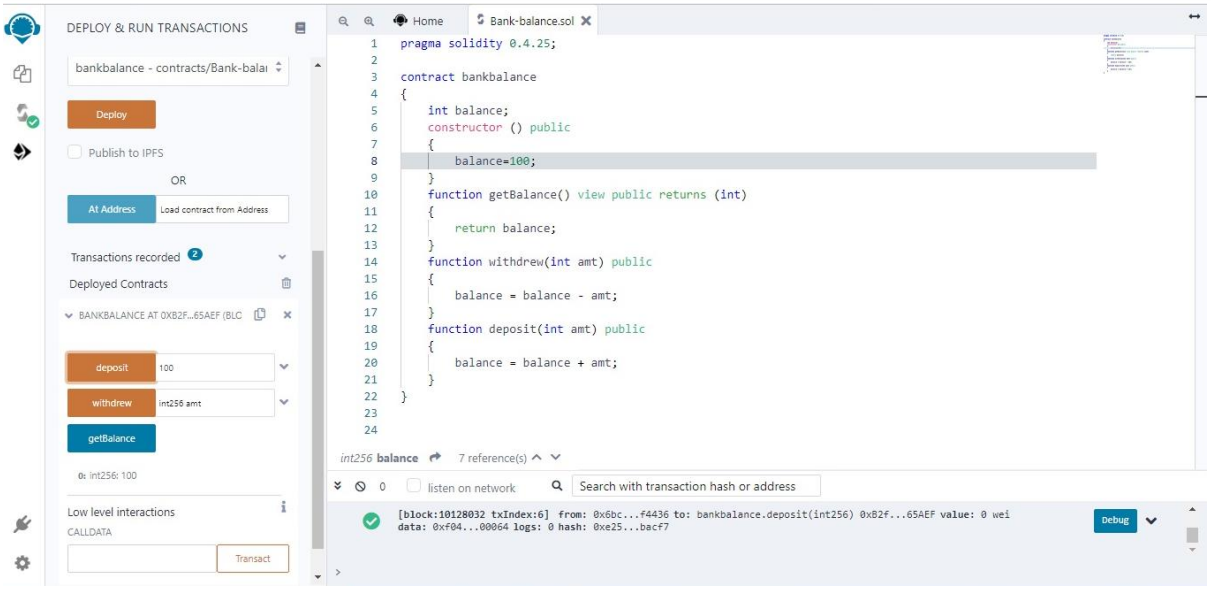
و تراکنش ها بصورت رمزنگاری شده با بلاک چین مورد استفاده قرار می گیرد. همچنین Metamax با ایجاد یک آدرس منحصر بفرد، ماهیت وجودی ما را با بلاک چین بصورت یکتا برقرار می کند. یعنی احراز هویت ما با بلاک چین توسط همین آدرس صورت می گیرد و در نقل و انتقال پول هم از همین آدرس استفاده می شود. ما برای تحقیق انجام داده شده از Rinkeby Test Network یعنی شبکه آزمایشی اتریوم که امکان آزمایش و توسعه بلاک چین را قبل از استقرار در Mainnet، یا شبکه اصلی اتریوم، فراهم می‌کند، استفاده کرده ایم [26].

۶. روش انجام تحقیق

ابتدا در محیط Remix IDE و به زبان Solidity یک برنامه ساده بنام Bank-balance.sol می نویسیم. کدهای قرارداد هوشمند ما بهمراه پرداخت هزینه توسط Metamax در شکل ۲ آمده است. و آنرا Compile و Deploy می کنیم با این کار بلاک چین ما بنام Bank-balance ایجاد می شود و با آدرس خاص خود که توسط آدرس API ، در محیط مجازی EVM قابل دسترسی می باشد (این آدرس یکتا و خاص است) ، شناسائی می شود. برای ارتباط با صفحه HTML نوشته و آماده شده ما ، بلاک چین بر بستر Injected Web3 ، Compile و Deploy شده است (شکل ۳).

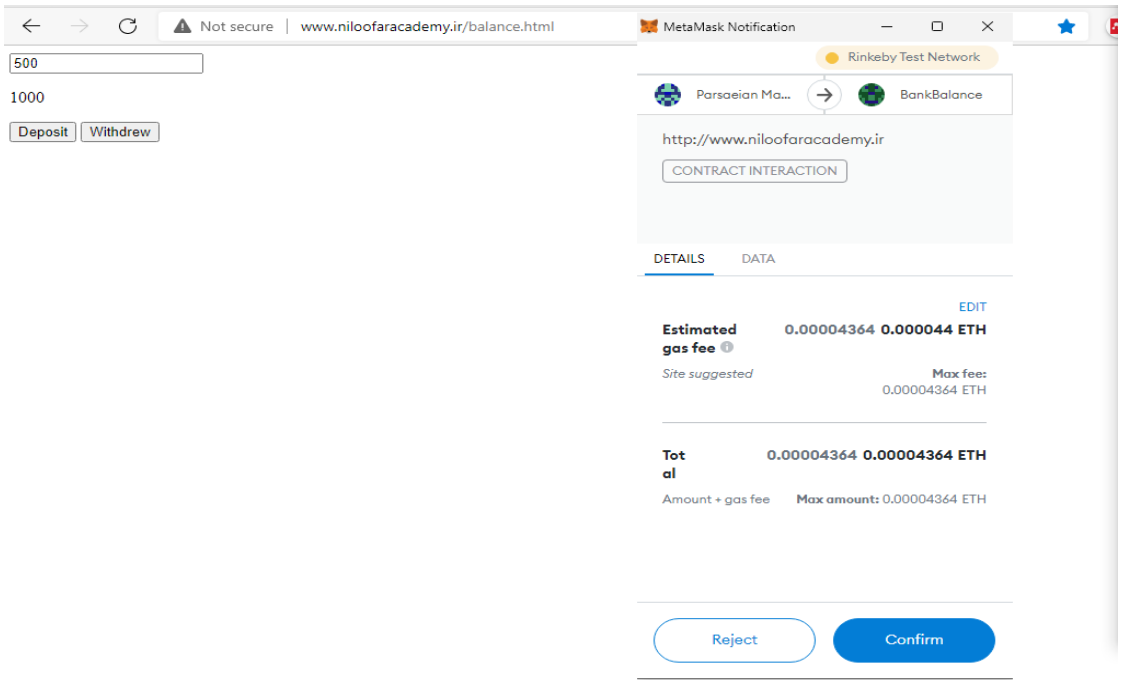


شکل ۲ نمایش همزمان قرارداد هوشمند و کیف پول متامکس قبل از کامپایل قرارداد هوشمند



شکل ۳ بلاک چین ایجاد شده در محیط EVM

این بلاک چین ابتدا مقدار موجودی balance را نمایش می دهد و مقدار جدید وارد شده amt توسط کاربر را می گیرد و بسته به فرمان های Deposit و یا Withdrew عمل جمع یا کم کردن مقدار وارد شده با مقدار balance را انجام و سپس مقدار جدید را در متغیر موجودی یا balance قرارمی دهد. (ارتباط با این بلاک چین توسط یک فایل نوشته شده به زبان HTML و استفاده از کدهای برنامه نویسی برای ارتباط های Client/Server همانطور که در بخش ۵ بیان شده است، مهیا شده است) شکل ۴ نمونه صفحه وب کاربر را نشان می دهد.



شکل ۴ نمونه صفحه وب کاربر برای تعامل با بلاک چین

همانطور که می بینیم بلاک چین ما توسط یک صفحه وب قابل دسترسی برای تمامی اعضای خاص خود می باشد. حال برای درک بهتر و اهمیت موضوع در نظر بگیرید که در این بلاک چین (Bank-balance) ، درخواستی برای کم کردن از حساب مشتری یا balance ارسال می شود (توسط فردی در یک مکان دیگر) و همزمان تقاضائی برای انتقال مبلغی از همان حساب یعنی از موجودی balance به حساب دیگری صادر می شود (توسط فردی غیر از فرد اول و در مکان دیگر). واضح است که اگر کنترلی برای انجام این تقاضا ها نباشد بدلیل خاصیت توزیع شده در بلاک چین، میزان موجودی هیچ وقت درست نخواهد بود. یعنی اگر مقدار اولیه موجودی ۵۰۰ واحد باشد و همزمان قرار باشد ۲۰۰ واحد با فراخوانی اول کسر شود (پس در آن لحظه موجودی باید ۳۰۰ باشد) در حالی که در فراخوان دوم درخواست انتقال ۴۰۰ واحدی هم رسیده است، (این بدان معنی است که عملا مشتری موجودی قابل برداشت ۴۰۰ واحدی ندارد بلکه موجودی او ۳۰۰ واحد است) ، دلیلش این است که هنوز فرمان اول کامل نشده است، و درنتیجه هنوز مقدار balance همان ۵۰۰ واحد در نظر گرفته می شود و عمل انتقال انجام می شود. پس این آسیب پذیری خطرناک وجود دارد. بدین صورت که وقتی فرمان دوم ، همزمان یا درخلال کار فرمان اول فراخوانی شود ، برنامه برای فرمان دوم هم مقدار موجودی را ۵۰۰ واحد در نظر می گیرد و عمل انتقال را انجام می دهد. با توجه به ماهیت بلاک چین که قابل دسترسی برای همه اعضاء می باشد این آسیب پذیری زیان بار و اثر گذار است. در شکل ۴ نحوه تعامل بین فایل وب و بلاک چین نشان داده شده است. (در بلاک چین برای هر تراکنش مبلغی از کیف پول ما بابت هزینه کسر می شود).

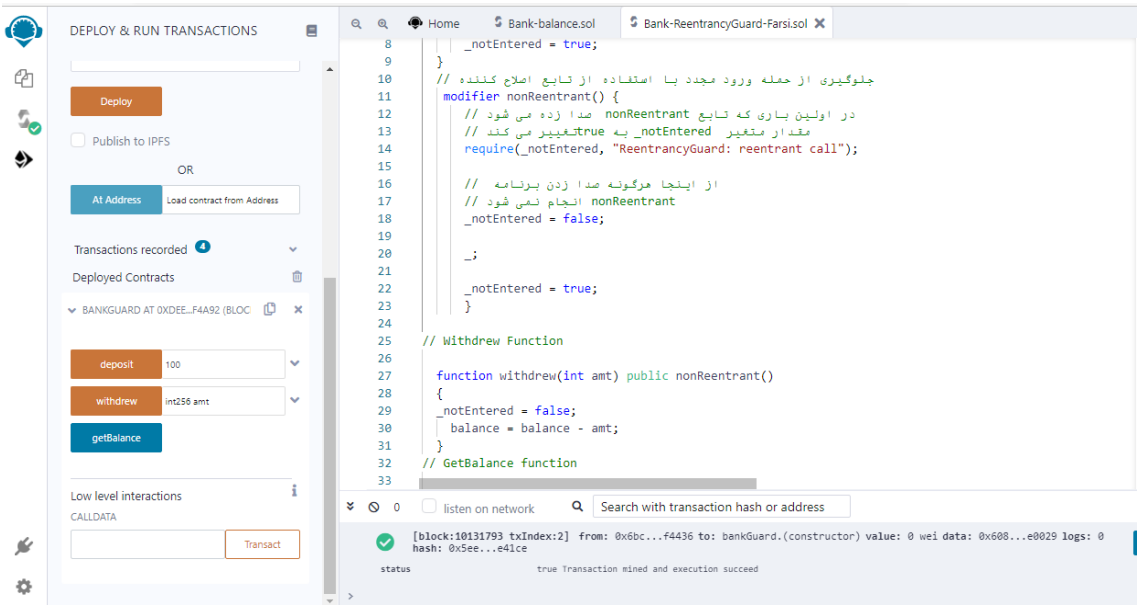
واضح است که هر تراکنش در این برنامه و برنامه های شبیه به این با توجه به ماهیت توزیع شده بلاکچین ، شرایط بوجود آمدن قراردادهای هوشمند، و زمان بر بودن انجام هر تراکنش (بخاطر ماهیت توزیع شده بلاک چین) بسیار آسیب پذیر هستند.

در راهکار پیشنهادی ما یک برنامه نویس می تواند به راحتی و بسته به نیاز خود جهت جلوگیری از حملات ورود مجدد در تمامی برنامه های پیچیده ای که دارد، از توابع اصلاح کننده یا Function Modifier ها استفاده کند. این راهکار ۱) برای محدود کردن دسترسی و ۲) برای اعتبار سنجی ورودی ها (فرمان) استفاده می شود.

در راهکار پیشنهادی ، ما با تعریف یک متغیر منطقی (logic) بنام _notEnterd برای محدود کردن دفعات فرا خوانی در قرارداد هوشمند و همچنین با تعریف تابع اصلاح کننده nonReentrant برای اولویت فرمان ها ، از حمله ورود مجدد جلوگیری می کنیم. بدین صورت که با تغییر مقدار متغیر _notEntered از true به false در زمان فراخوانی توابع Deposit و Withdrew و وجود Function Modifier در حین اجرای قرارداد هوشمند، محدودیت ایجاد می کنیم. در راهکار ارائه شده پس از فراخوانی تابع اول مقدار متغیر منطقی ما عوض می شود و تا زمانیکه که کار آن تابع به پایان نرسیده باشد قرارداد هوشمند ما هیچ فرمان اجرائی دیگری را قبول نمی کند ، و اگر فرمانی فرا خوانده شد ، منتظر می ماند تا پایان کار تابع فراخوانده شده اول فرا برسد و سپس شروع به کار کند.

در این راهکار ما با داشتن متغیر منطقی محدودیت دسترسی ایجاد می کنیم و با Function Modifier کار معتبر بودن ماهیت فرمان در حالی که فرمان قبلی هنوز در حال اجرا است را بررسی می کنیم. پس با توجه به تمهیدات انجام شده، مدامیکه فرمان اول کارش را به پایان نرسانده باشد، هیچ فرمانی را از قراردادهای هوشمند نخواهد پذیرفت.

بعلاوه پس از پایان کار هر یک از توابع فراخوانی شده (Deposit, Withdrew) توسط ما ، مقدار متغیر منطقی به مقدار اولیه عوض می شود تا برنامه امکان اجرای مجدد را داشته باشد. چگونگی کدهای قرارداد هوشمند بهمراه بلاک چین جدید (BankGuard) که در مقابل حمله ورود مجدد مقاوم است در شکل ۵ و ۶ نشان داده شده است.



شکل ۵ : کدهای قرارداد هوشمند که در مقابل حمله ورود مجدد مقاوم است

شکل ۶ نمونه صفحه وب کاربر در تعامل با بلاک چین که درمقابل حمله ورود مجدد مقاوم است

۷. بحث و نتیجه گیری

افزایش محبوبیت فن آوری بلاک چین بدون اشکالاتی مانند نقص های امنیتی، مسائل پروتکل های توافقی، و مقیاس پذیری نبوده است. اخیراً برخی از روش ها برای به حداقل رساندن بسیاری از نقص ها و آسیب پذیری ها پیشنهاد شده اند، اگرچه گسترده هستند. ولی هنوز تا پذیرش روش‌های منحصر به‌فرد فاصله زیادی وجود دارد.

باتوجه به پرخطر بودن حملات ورود مجدد در قراردادهای هوشمند. ما با ارائه راهکاری دقیق و کارآمد جهت مقابله با این نوع حملات اقدام کردیم. راهکار ما از نظرکد نویسی بسیار ساده ، دقیق و بدلیل کم حجم بودن از نظر بایت های بکار رفته (در زمان کامپایل قرارداد هوشمند) بسیار کاربردی می باشد. همچنین راهکار پیشنهادی ما قابل تعمیم به زبان های دیگر برنامه نویسی که در بخش ۲ شرح داده شده است و بر بستر بلاک چین Ethereum کار می کنند ، می باشد. ما نشان دادیم با درک بهتر از توانائی های Function Modifier ها و کاربرد دقیق آن در کد نویسی قرارداد هوشمند، به همراه بکارگیری متغیر های منطقی ، بصورت مطمئن می توانیم از حملات ورود مجدد در قرار دادهای هوشمند جلوگیری کنیم.

زمانی که با یک سیستم غیر متمرکز (Decentralized) مواجه هستیم، بدلیل همین خاصیت غیر متمرکز بودن، مشکلات همزمانی اجرای برنامه ها وجود دارد، ما برای این مشکل راه حل دقیقی جهت جلوگیری از حملات ورود مجدد ارائه کردیم که مبتنی بر اصل تجزیه و تحلیل دقیق سیستم و درک درست از کدهای برنامه نویسی به همراه چگونگی کاربرد صحیح آن کدها در قرارداد هوشمند است. این راهکار در تبیین یک قرارداد هوشمند بی نقص و مقاوم در برابر حمله ورود مجدد کاملا موثر است. ما در این مطالعه نشان دادیم که داشتن تصویری دقیق از تمام جزئیات یک کسب‌وکار همچنین تعیین شرایط دقیق برای پارامترهای قرارداد هوشمند از ضروریات یک قرارداد هوشمند در محیط بلاک چین است.

بعلاوه کسب و کارهایی می توانند بر اساس بلاک چین و قرارداد هوشمند ساخته و اجرا شوند که Algorithm Trust باشند یعنی بتوانیم الگوریتمی بی نقص از شرایط و ضوابط قراردادهای هوشمند ایجاد و بصورت دقیق برای پیاده سازی و اجرای یک قرارداد هوشمند از آن استفاده کنیم.

برنامه ها و کدهای برنامه نویسی نوشته شده توسط برنامه نویسان نیز باید عاری از خطا و دقیق تنظیم شده باشند تا از هر گونه شرایط پیش بینی نشده ای که می تواند منجر به خطا و آسیب پذیری در حین اجرای قرارداد هوشمند شود، جلوگیری کند. چنانچه ما با بکاربردن صحیح کدهای برنامه و تعریف متغیر های منطقی به این هدف رسیدیم.

۸. تحقیقات و جهت گیری های آینده

داشتن الگوی مبتنی بر یادگیری ماشین (Machine Learning) می تواند در آینده مفید باشد. یعنی با Supervised Learning، اطلاعات آسیب‌پذیری‌های موجود و شناخته شده را روی رایانه تعریف می‌کنیم و سپس می‌توانیم قبل از اجرای قرارداد هوشمند، آسیب‌پذیری‌ها را توسط ML آزمایش کنیم. قراردادهای هوشمند زمانی اجرا می‌شوند که از تمام آسیب‌پذیری‌هایی که توسط ML ارائه می‌شود در امان باشند.

اکنون در روش یاد گیری ماشین ، می توان با یادگیری بدون نظارت (unsupervised Learning) یعنی اینکه همه آسیب‌پذیری‌هائی که در زمان وقوع تعریف می‌شوند (حمله روز صفر) نیز اطلاعات را در مجموعه داده‌های یادگیری نظارت شده ذخیره کرد و سپس از آن برای حملات مشابه در آینده استفاده کرد. این روش ما را در برابر حملات روز صفر مقاوم می کند.

۹. نتیجه گیری و کار در آینده

در زیر پیشنهاداتی وجود دارد که می تواند امنیت در برابر آسیب پذیری های قراردادهای هوشمند را بهبود بخشد:

الف) تصویر واضح و جامع از کسب و کار (روش های عملیاتی) داشته باشیم.

ب) شرایط قراردادها واضح و بدون تعارض باشند.

ج) ایجاد متغیر های منطقی برای پارامترهای مورد نظر.

د) پیش بینی کردن خطاها و گسست داده ها .

منابع

[1] "What is Blockchain Technology - IBM Blockchain | IBM." https://www.ibm.com/topics/what-is-blockchain (accessed Jul. 13, 2021).

[2] S. Nakamoto, "Bitcoin : A Peer-to-Peer Electronic Cash System," pp. 1–9.

[3] N. T. Courtois, M. Grajek, and R. Naik, "The Unreasonable Fundamental Incertitudes Behind Bitcoin Mining," no. March, 2014.

[4] "Top 10 Cryptocurrencies in Value in May 2021." https://www.analyticsinsight.net/top-10-cryptocurrencies-in-value-in-may-2021/ (accessed Dec. 07, 2021).

[5] "Proof-of-work (PoW) | ethereum.org." https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/ (accessed Jul. 12, 2021).

[6] "Proof-of-stake (PoS) | ethereum.org." https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/ (accessed Jul. 18, 2021).

[7] Y. Sun, B. Yan, Y. Yao, and J. Yu, "ScienceDirect DT-DPoS: A Delegated Proof of Stake Consensus Algorithm with Dynamic Trust," *Procedia Comput. Sci.*, vol. 187, pp. 371–376, 2021, doi: 10.1016/j.procs.2021.04.113.

[8] "What are smart contracts on blockchain | IBM." https://www.ibm.com/topics/smart-contracts (accessed Jul. 13, 2021).

[9] "Best Programming Languages to Build Smart Contracts." https://www.blockchain-council.org/blockchain/best-programming-languages-to-build-smart-contracts/ (accessed Jul. 12, 2021).

[10] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, and F. Y. Wang, "An Overview of Smart Contract: Architecture, Applications, and Future Trends," *IEEE Intell. Veh. Symp. Proc.*, vol. 2018-June, no. Iv, pp. 108–113, 2018, doi: 10.1109/IVS.2018.8500488.

[11] G. Destefanis, M. Marchesi, M. Ortu, and R. Tonelli, "Smart Contracts Vulnerabilities : A Call for Blockchain Software Engineering ?," no. March, 2018, doi: 10.1109/IWBOSE.2018.8327567.

[12] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," *Proc. - Int. Conf. Softw. Eng.*, pp. 65–68, 2018, doi: 10.1145/3183440.3183495.

[13] Y. Marcus, E. Heilman, and S. Goldberg, "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network.," *IACR Cryptol. ePrint Arch.*, vol. 2018, no. January, p. 236, 2018.

[14] X. Wang *et al.*, "Attack and Defence of Ethereum Remote APIs," *2018 IEEE Globecom Work. GC Wkshps 2018 - Proc.*, pp. 1–6, 2019, doi: 10.1109/GLOCOMW.2018.8644498.

[15] M. Rodler, W. Li, G. O. Karame, and L. Davi, "Sereum: Protecting Existing Smart Contracts Against Re-Entrancy Attacks," no. February, 2019, doi: 10.14722/ndss.2019.23413.

[16] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "EASYFLOW: Keep ethereum away from overflow," *Proc. - 2019 IEEE/ACM 41st Int. Conf. Softw. Eng. Companion, ICSE-Companion 2019*, no. April 2018, pp. 23–26, 2019, doi: 10.1109/ICSE-Companion.2019.00029.

[17] S. Sayeed, H. Marco-Gisbert, and T. Caira, "Smart Contract: Attacks and Protections," *IEEE Access*, vol. 8, pp. 24416–24427, 2020, doi: 10.1109/ACCESS.2020.2970495.

[18] B. Prasad and S. Ramachandram, "SMART CONTRACTS : VULNERABILITIES AND," vol. 11, no. 8, pp. 901–922, 2020, doi: 10.34218/IJARET.11.8.2020.089.

[19] A. Alkhalifah, A. Ng, P. A. Watters, and A. S. M. Kayes, "A Mechanism to Detect and Prevent Ethereum Blockchain Smart Contract Reentrancy Attacks," *Front. Comput. Sci.*, vol. 3, no. February, pp. 1–15, 2021, doi: 10.3389/fcomp.2021.598780.

[20] "Remix - Ethereum IDE." https://remix.ethereum.org (accessed Feb. 05, 2022).

[21] "Visual Studio Code - Code Editing. Redefined." https://code.visualstudio.com/ (accessed Feb. 05, 2022).

[22] "About npm | npm Docs." https://docs.npmjs.com/about-npm (accessed Feb. 02, 2022).

[23] "Node.js." https://nodejs.org/en/ (accessed Feb. 02, 2022).

[24] "web3.js - Ethereum JavaScript API — web3.js 1.0.0 documentation." https://web3js.readthedocs.io/en/v1.2.9/ (accessed Feb. 02, 2022).

[25] "A crypto wallet & gateway to blockchain apps | MetaMask." https://metamask.io/ (accessed Feb. 02, 2022).

[26] "Rinkeby: Network Dashboard." https://www.rinkeby.io/#stats (accessed Feb. 02, 2022).