Slide 1:

Hey guys, my name is Rotem Brillant and ill be presenting to you guys the paper that was written by Daniel Perez and Ben Livshits and it's title is Smart Contract Vulnerabilities: Vulnerable Does Not Imply Exploited.

Slide 2:

So as the article title may suggest, one of the main questions of this article is "Does vulnerable mean exploited?". As in, does it necessarily mean that a contract flagged as vulnerable has actually been exploited? Has it's Ether (which is the currency used) been stolen? The second question this article aims to answer is "how much Ether has been Exploited?" and give an upper bound to the amount.

Slide 3:

So, what do other papers say? This paper chooses to look at the results of 6 other recent papers that have flagged 23,327 contracts as vulnerable. Out of those contracts the papers derived that a total of 600 million dollars is at risk, which is of course quite a big sum!

Slide 4:

But don’t worry ladies and gentlemen, this article will try to convince us that this amount is exaggerated. We will go through the different methodology used in this paper, and explain what tool was constructed to get to our desired result, which is that a lot less Ether is at stake.

Slide 5:

Before we get into the details of the article, we must first be able to understand what Ethereum is.

Slide 6:

So this is Vitalik Buterin, a Russian Canadian programmer that was really into the new field of cryptocurrency which was really booming since the earlier launch of Bitcoin. As you can see, at the year of 2013 he has already thought of the idea at the young age of 19. Only 2 years later Ethereum was launched

Slide 7:

So what are the Three Pillars that make any blockchain cryptocurrency unique?

Decentralization: The data is recorded and stored on multiple devices and no one entity controls the data record or storage process.

Transparency: The way in which transactions are recorded on a ledger that is available for everyone to see and that is saved on a network of computers around the world.

Immutability: The data that is stored on the blockchain cannot be changed, forged or altered thanks to cryptography and the blockchain hashing process.

So what's the difference between Ethereum and the rest of the rest of the cryptocurrencies?

Slide 8:

Well, Ethereum is more than just a currency. It has the capability to do much more than just pass around money with no middleman. It has a currency called Ether, Decentralized Autonomous Organizations, Decentralized Applications and Smart Contracts, which will be our focus. DAOs are essentially a board of investors, such that the more you invested the more you can influence on voting which projects the organization will invest in, which could

be anything of course. Dapps are essentially like any other app you guys know, only that it runs on the Ethereum Network such as games, gambling and more. Both DAO's and Dapps are implemented using Smart Contracts. Ether is the cryptocurrency used on Ethereum, and is essentially the fuel that runs this whole network.

And now a little more on Smart Contracts

Slide 10:

So these are some of the important features of Smart Contracts.

So these contracts are written in actual code, specifically the main 2 coding languages used are Solidity which is similar to JavaScript and Vyper which is a little bit of everything but mainly takes inspiration from python.

They are immutable, meaning the moment I write a contract and upload it to the Ethereum Network, it cannot be changed, meaning I cannot change the code in the contract.

The contracts self-execute. This is pretty intuitive to understand. Like any other coding projects you guys have done, once certain conditions hold then the code is executed, and this cannot be stopped by anyone

Like any other account, smart contracts also have an address, they have a balance i.e. they can hold Ether and they have their own storage.

And of course, like any other account, any transaction that is done with a contract is also recorded to the blockchain

Slide 11:

So here is just an example of a very simple Wallet contract. It has a constructor which is called only once when the contract is made. It has local variables like owner, and so you can see that when the Wallet is created the owner is the one who created the Wallet contract. It has a few other simple functions like withdraw and deposit and a change owner. We also have a special function here called a modifier which serves for certain checks, so for example in order to change the owner of the contract, the only_owner modifier is called before changeOwner to validate that the owner is actually calling. Only then will changeOwner execute it's code.

So really you can already see that in Smart Contracts it really is as they say, Code is Law. You can also see that you should be a very good and cautious programmer if you are going to make your contract not vulnerable.

Slide 12:

So, another important component of Ethereum that needs mentioning is the Ethereum Virtual Machine, or EVM. It’s a stack-based VM which has most of the familiar stack instructions you guys know and also other instructions related to Ethereum. So here we have an example of some random contract written in Solidity and how the code looks like after compilation.

Slide 14:

So in summary about smart contracts, lets have a look at their life cycle. So you have a seller who wrote a contract for some arbitrary service, and a buyer which went over the contract and likes what its offering. Certain events occur which cause an execution of the relevant code in

the contract which causes transactions between the relevant parties, and this is all recorded on the blockchain.

Ok, so now that we have the right amount of knowledge on Ethereum and Smart Contracts, lets get back to the article and discuss…

Slide 15:

How do we even find vulnerabilities? What type of tools are there, and what was used in those 6 other papers we mentioned in the beginning?

Slide 16:

So the main tools used today to find vulnerabilities are called Static Code Analysis and Dynamic Code Analysis tools. The main difference between the two is that Static code tools analyze the code itself, looking for certain patterns that can lead to vulnerabilities, while Dynamic Code tools give actual input to the code and check if a vulnerability has been triggered. Specifically for Ethereum, static code analysis tools are mainly used to check for vulnerabilities and there are quite a few of them. Some check the EVM code, others the Solidity code. The tool used in this paper looks on the EVM code to do it's thing, which will be explained later on. The names you see before you are just the 6 tools created by the 6 other papers I have mentioned.

Slide 17:

So, the reasonable question to ask is what's the difference between the tool my paper presents, and the 6 other tools represented by those papers? What makes this paper unique to others?

Slide 18:

Well the main difference is the methodology. Lets go through what their plan was. So first, for each contract they retrieved all the contract transaction data. This is of course recorded on the blockchain. Next, they extracted all the execution traces of the transactions. Then, they replayed the transactions, and encoded them using their new tool. Finally, the end result is of course, we find exploited contracts! Yay!

Slide 19:

Finally, we have here all the vulnerabilities that are checked by this paper. Let's simply explain each one of them.

Slide 20:

Re-Entrancy.

Slide 21:

So we cannot talk about Re-Entrancy without mentioning the DAO Hack which is probably the most considerable hack that happened on the Ethereum since it's launch. The investors were able to raise 150 million USD, and 60 million USD worth of Ether was stolen using this attack. This hack was so controversial that it caused the Ethereum community to decide on a "hard-fork" which created Ethereum and Ethereum Classic. The Ethereum blockchain essentially decided to "rewrite" the blockchain history such that the hack never happened, while Ethereum Classic remains with the original blockchain.

Slide 22:

Unhandled Exceptions.

Slide 23:

Locked Ether, explain animation. An important thing that I should also mention is of course that a contract can be "destroyed" in the sense that it is just not available anymore to be called, but it is still on the blockchain and is not deleted.

Slide 24:

So again, we have here another significant hack that exploited this vulnerability and caused 280 million USD worth of Ethereum to be frozen. What happened is that the Parity Wallet used an outer library contract to implement important functionalities, such as to withdraw Ether. An attacker saw a vulnerability in the library contract which enabled him to be the owner of that contract, and use the selfdestruct function to make that contract unusable. This caused of course, that all the wallets owned by individuals were unable to withdraw any of their money.

Slide 25:

Integer Overflow

Slide 26:

Unrestricted Action. Certain actions are of course prohibited to only certain individuals, such as changing the owner, using selfdestruct, writing to the storage and many more. There should be verification for anyone of these instructions.

Slide 27:

Transaction Order Dependency. So essentially this is some sort of a race condition attack i.e. the order of which certain transactions are processed can change the final result. In order for this attack to happen there needs to be

2 or more transactions with the same contract that are on the same block. So lets say for example I have a contract selling something for 100 Eth. One transaction is me wanting to buy this product for 100 Eth, and another transaction by the owner which he changes the price of the product to 150. So if my transaction is processed first and only then the owners transaction, everyone is happy. But you guys can already tell that if my transaction was processed second, then I paid more than what intended.

Slide 28:

So now that we defined the vulnerabilities and gave some examples, lets dive into the details…

Slide 29:

First, lets look at the database that was used. We have 821,219 contracts that were covered by 6 other papers that were mentioned before. 23,327 of those contracts has been flagged as vulnerable by at least 1 of the vulnerabilities mentioned before, by at least 1 of the tools used in the other papers.

Slide 30:

Finally, lets take a look at the tool used in this paper. It is a first-order logic, it has variables, addresses and is comprised of Facts, Rules and Queries.

Slide 31:

So here are the facts, rules and queries. I am not going to explain each and everyone here. I am only showing you guys this to get a sense of how this looks.

Slide 32:

So now that I have presented to you guys all the components that we need to know in order to understand the paper, let's do a little recap to put it all together.

Slide 33:

So we retrieve for every contract it's transaction history, extracts the execution traces, replay the transactions. While doing so, we encode these transactions using our new tool and record facts. We insert a query for a certain vulnerability, and then hope that we find an exploited contract.

Slide 34:

Ok, lets dive deeper into the details of each vulnerability and how our new tool can find them.

Slide 35:

So these are the facts and rules that are relevant but I wont get into any of them. Lets just understand the query which all are pretty intuitive and easy to understand. So since we are looking for recursive calls we are looking for a1 who called a2, and then of course a call from a2 to a1 such that a1 != a2 which exactly resembles a recursive call. We collect the relevant facts and check if the query is valid or not

Slide 36:

So the main issue with this method is that we have no way of assessing if a re-entrant call is malicious or not, which can lead to false positives.

Slide 37:

So for Unhandled exceptions the focus is on methods which send Ether for example "send". So looking at the

query, we are looking for a call result that returned 0, and if the variable v was used in a condition or not. So we search for both JUMPI and CALL instructions, record the facts and check if the query is valid.

Slide 38:

So the main issue with this method is that now we are recording all failed calls, and not just ether sending calls, which can lead to false positives. Secondly, if a call result is used but it isn’t enough to prevent the exploit, we will get false negatives.

Slide 39:

For locked Ether the paper focuses on the case where 1 contract relies on another which does not exist anymore. An important instruction that is important to the attack is DELEGATECALL which is when 1 contract calls another while granting access to it's storage. Essentially its like doing copy paste of contract B's relevant code to contract A. Another important thing to notice is the behavior of the EVM when a contract calls another contract that does not exist anymore. Such an occurrence does not lead to a failure, and the next line of code is executed as if nothing happened.

Slide 40:

So looking at the query, the thing that interests us is that there was a call from contract A when the PC was i1, and that the call exit when PC was i2 (which is where we did "return") was actually the next instruction, i.e. that i1 + 1 = i2. We search for DELEGATECALL instructions, record the facts and check the query.

Slide 41:

So actually, for this vulnerability that are no issues, but we must remember that this is only for this very specific case.

Slide 42:

Moving onto transaction order dependency, here are 2 very straightforward instructions that we need to know. The paper is looking at the case where one transaction is writing into the storage and another transaction is reading from the same place in the storage.

Slide 43:

Looking at the query, we are looking for the case mentioned before, both are in the same block doing different operations on the storage with the same key and of course that the transactions are different. So we look for SSTORE and SLOAD instructions, record the facts and check if the query is valid.

Slide 44:

So the issues for this method is that we don’t always know how the storage is handled, and who can or cannot read/write from it, which can lead to false positives.

Slide 45:

So for the last 2 vulnerabilities which are the most complex, lets just understand the queries and ill try to simplify the rest. So for integer overflow query we can see is quite intuitive. For a variable v we will check the actual result vs the expected result of v and check if they are different.

Slide 46:

So first, for every variable, we will try to extract its size and value and record them as facts. In the EVM, we cannot tell the difference between different types, and so the tool relies on certain heuristics of the way Solidity is compiled to EVM code. Next, we use the data we extracted from the different variables and compute the expected value for all arithmetic operations in the execution trace. Finally we compare between the expected and actual results.

Slide 47:

So what are the issues with the way we check for this vulnerability? The dependency on a specific heuristic can lead to both false positives and negatives.

Slide 48:

Ok, for the final vulnerability we have Unrestricted Action. The tool tried to check for 3 different unrestricted actions:

Unrestricted Calls which was focused specifically on unrestricted calls to selfdestruct.

Unrestricted Writes which is exactly as it implies.

Finally, for Code Injection the tool checks whether the passed data influenced the address called by DELEGATECALL or CALLCODE.

Slide 49:

So here are the rules and the facts, and these are just partial since there isn’t enough room on the slide, but lets just understand the query. We have 2 cases such that the query can be valid. The first one is pretty easy to understand, who called the selfdestruct and was he checked? The second is who (the variable) was used to call the restricted instruction(restricted_inst(v)), was he

checked (!called_checked(v)), what was the data passed and has it been manipulated?

Slide 50:

Similar to before, we gather facts and we search for JUMPI instructions to see if certain variables were used to influence a condition. The query that was asked for the selfdestruct is pretty easy. Who called the selfdestruct, and has he been checked? For the other 2 cases, the query is also quite intuitive. Who carried out the restricted instruction? Has he been checked? What was that data that was sent and has it been manipulated in any way?

Slide 51:

The issues that can arise from such a query is pretty obvious. What if certain contracts don’t have writing restrictions to the storage? This can of course lead to false positives. Also, what if a condition is written incorrectly, such that the contract was still exploited even if certain checks were made? This can of course lead to false negatives.

Slide 52:

Ok, after looking into all the different vulnerabilities and how the writers of the paper were able to find them, lets see what the results are.

Slide 53:

Explain the result table.

So after looking into the result table, which gives us a very good result and confirms the writer's hypothesis that much less Ether has been exploited, what did the writers do to validate their result?

Slide 54:

Firstly, for every type of vulnerability, the writers manually checked the top "valuable" contracts i.e., those that had the most Ether exploited to confirm that contract was indeed exploited. Furthermore, they also performed Sanity checks of their tool on both the Dao and the Parity Wallet hacks (which were not part of the database) and got similar results to other referenced papers. Further analysis of the results reveals that there is a major difference between the "vulnerable" and "exploited" Ether, and the writers explain this.

Slide 55:

First of all, the writers find that from the database that was used, the top 10 contracts hold 95% of the total Ether. This suggests that "top" contracts are less exploitable i.e., coded better and safer and if they were marked as vulnerable by other papers it was either a false positive or are not exploitable in practice. Furthermore, the distribution of Ether among contracts in the used database generalizes with the whole Ethereum blockchain, meaning most Ether is concentrated in certain contracts.

Slide 56:

So here is just what we were talking about. Only 2000 contracts in the whole Database had Ether, and as you can see most of them had less than 1 Ether

Slide 57:

So, what was the conclusion of the paper? No more than 463 flagged vulnerable contracts have actually been exploited. This follows with the result that at most only 0.27% of Ether was exploited, out of the 3 million Ether

thought to be at risk. And of course, that most of the Ether is held by a small number of contracts which seem to be safe.

Slide 58:

I would say those are pretty good results

Slide 59:

So my critique for this paper would be divided into 2 topics:

1. Readability and thoroughness (יסודיות)
2. Faults of the Tool

Readability and thoroughness:

First of all, the paper could use some examples for the different attacks and explanations of different EVM instructions. Many papers in this field of interest have such examples and explanations. Furthermore, some facts and rules were very vague and hard to understand how they were implemented in finding the exploited contracts.

Faults of the tool:

The tool that was comprised by the authors of this paper relies on Solidity specific compilation by the EVM, such as Integer Overflow and Unhandled Exceptions, while smart contracts can be written in numerous languages, Vyper being one of them

Slide 60:

A look to the future in implementing and using what was brought about in this paper I think is pretty limited, though can further establish the trust people can have in Ethereum. This can be extending the tool to different vulnerabilities which there are many of, only a few listed here in this slide.

Slide 61:

All in all, this paper was built very well. From the main question this article tries to answer which is how much Ether has actually been exploited, to the new methodology that was presented that has never been implemented before. It increases the trust in the blockchain technology and specifically in Ethereum contracts such that all you need to do is write safe code.