Decentralized Threat Detection Bots.

Research and development.

Jonathan Alexander

CTO OpenZeppelin, co-founder Forta

Contributors to this presentation.

Thank you to the following whose work is cited in this presentation:

- Christian Seifert, researcher in residence, Forta
- Mariko Wakabayashi, lead ML engineer, OpenZeppelin
- Dario Lo Buglio, security researcher, OpenZeppelin
- Artem Kovalchuk, Vyashceslav Trushkov, and Soptq, independent researchers
- Development teams at Nethermind and LimeChain

Section 1

Background.

Runtime monitoring and threat detection.

Multiple leading security audit firms (OpenZeppelin, ChainSecurity, Halborn, Mixbytes) are beginning to make recommendations on post-deployment smart contract monitoring. Recommendations to monitor include:

- Protocol assumptions and invariants
- State of critical protocol variables
- Known protocol risks that have been considered acceptable
- Privileged protocol functionality and transfers of privilege
- On-chain / off-chain / cross-chain synchronization (oracles, bridges)
- External contracts that protocol relies on or is exposed to
- Identified user and protocol attack patterns

Try to catch the knowns, the known unknowns, and the unknown unknowns

Runtime monitoring and threat detection.

Multiple leading security audit firms (OpenZeppelin, ChainSecurity, Halborn, Mixbytes) are beginning to make recommendations on post-deployment smart contract monitoring. Recommendations to monitor include:

- On-chain / off-chain / cross-chain synchronization (oracles, bridges)
- External contracts that protocol relies on or is exposed to
- Identified user and protocol attack patterns

Very challenging for protocol teams to implement effectively by themselves

Name		Node operators	Detection Bots
1	Ethereum	1051	691
2	Polygon	298	254
3	BSC	161	122
4	Avalanche	143	100
5	Arbitrum	134	70
6	Optimism	125	74
7	Fantom	57	45

Forta is designed to be a community monitoring and threat detection platform.

- Test network 2021, public network 2022
- Non-profit Forta Foundation backed by a16z, Blockchain Capital, and many others
- Permissionless node running, security staking
- Permissionless bot deployment, node redundancy
- Community services for alert subscriptions and notifications
- Governance: Council at non-profit Foundation, Forta Proposal Process, Snapshot voting

github.com/forta-network app.forta.network explorer.forta.network

Section 2

Research and observations on attacks.

Smart contract attack stages.

Funding

- New account
- Mixer, CEX or bridge transfers

Preparation

- Contract deployment
- Token impersonation
- Privilege grants / transfers
- Sleep minting
- Ice phishing

Exploitation

- Flash loans
- Flashbots
- Re-entrancy
- Minting
- Anomalous balance or price changes
- Transfers

Laundering

- Mixer, CEX or bridge deposits
- Wash trading

103 of 181 (57%) of DeFi attacks in last 3 years are non-atomic, meaning they could have been identified as progressing through the above stages, and even after the first exploit TX there was rescue time available (https://arxiv.org/pdf/2208.13035.pdf)

Use heuristics to associate and track attacker accounts through stages.

Attackers may do funding through a mixer to account A but then transfer funds to account B and carry out preparation and an exploit from there. Using heuristic-based approaches, such as a **connected component graph algorithm**, accounts that interact can be grouped into clusters, and then stages of an attack can be tracked for a given cluster.

Attackers often use more than one account.

In >40% of attacks, the attacker deploys a smart contract to execute the exploit.

Attack contracts differ from benign contracts.

Using SVM (support vector machine) classification on the top 100 opcode function signatures of 10,000 smart contracts sourced from Luabase, and 155 EOAs tagged with "exploit" in Etherscan, and a 70/30 split of training/testing data, the classifier was able to classify **98% of benign** contracts and **81% of malicious** contracts.

	Benign Prediction	Malicious Prediction
Benign	2,946	52
Malicious	4	17

Bytecode analysis can reveal attack patterns.

Used the Term Frequency - Inverse Document Frequency (TF-IDF) technique from NLP which extracts opcodes in unigrams, bigrams, trigrams and 4-grams:

- example unigram: PUSH1
- example 4-gram: PUSH1 MSTORE PUSH1 CALLDATASIZE

Trained on 12,864 benign contracts and 103 malicious contracts, fed into logistic regression with stochastic gradient descent (SGD) classifier.

Identified malicious contracts with **88% precision** and **59% recall**. Specifically this technique identified:

- Wintermute 2 Exploit: '0x0248F752802B2cfB4373cc0c3bC3964429385c26'
- Audius Exploit: '0xbdbB5945f252bc3466A319CDcC3EE8056bf2e569'
  Inverse Finance Exploit: '0xf508c58ce37ce40a40997C715075172691F92e2D'

For data see https://github.com/forta-network/labelled-datasets

Exploits executed via smart contracts follow similar patterns.

"Fraud" attacks produce detectable on-chain patterns.

Heuristic-based analysis can detect fraud.

In **ice phishing** an attacker uses web2 phishing techniques to trick users into signing approval transactions giving the attacker control of tokens. A **heuristic-based technique** was used to detect multiple token approvals/transfers to a single EOA along with other heuristics to reduce false positives. During a 1 week period in Sept 2022, 21 ice phishing attacks were identified from phishing reports filed on ChainAbuse for Ethereum mainnet. Of these, the heuristic technique identified 12 of the 21 attacks for **57% recall** with **95% precision**.

Section 3

Threat detection bot techniques.

Forta bot development (JS, Python).

Handlers

```
type Initialize = () => Promise<void>
type HandleTransaction = (txEvent: TransactionEvent) => Promise<Finding[]>
type HandleBlock = (blockEvent: BlockEvent) => Promise<Finding[]>
```

Functions

- TransactionEvent.filterLog
- TransactionEvent.filterFunction
- getJsonRpcUrl
- getEthersProvider
- getTransactionReceipt
- getAlerts
- fetchJwt

Return

- Finding

Test and Integration Helpers

- createBlockEvent
- createTransactionEvent
- verifyJwt
- decodeJwt

Test Tools

- CLI Run
- Forta scan node local mode

Bot Technique: Multiple bots working together in a group to track attack stages.

Atomic detection, account clustering, and alert pattern matching.

Relevant bots:

Suspicious contract creation
https://github.com/forta-network/starter-kits/tree/main/suspicious-contract-creation-py

Social engineering (contract spoofing)
https://github.com/forta-network/starter-kits/tree/main/social-eng-contract-py

Ice phishing
https://github.com/LimeChain/forta-starter-kits/tree/main/ice-phishing

Large transfer
https://github.com/forta-network/starter-kits/tree/main/large-transfer-out-py

Money laundering
https://github.com/forta-network/starter-kits/tree/main/money-laundering-tornado-cash-py

Entity (account) clustering
https://github.com/forta-network/starter-kits/tree/main/entity-cluster-bot

Alert combiner (alert pattern detector)
https://github.com/forta-network/starter-kits/tree/main/alert-combiner-py

Fork the chain in a bot and run simulation tests.

Relevant bots:

Dynamic liquidity testing (try withdrawals for top users)
https://github.com/NethermindEth/Forta-Agents/tree/main/Yearn-agents/InstantWithdrawal

Attack simulation (simulate newly deployed contracts)
https://github.com/Soptq/bot-attack-simulation

Attack simulation with fuzzing
https://github.com/kovart/forta-attack-simulation

Bot Technique: Simulate user TXs or contract executions to identify attacks or malicious contracts.

Deploying ML models in bots.

Serialize the model

```
import dill

with open('isolation_forest.pkl','wb') as f:
    dill.dump(model, f)
```

Add the model to the bot dockerfile

```
WORKDIR /app
COPY ./isolation_forest.pkl ./
```

Load model in the initialize handler

```
ML_MODEL = None

def initialize():
   global ML_MODEL
   logger.info('Start loading model')
   with open('isolation_forest.pkl', 'rb') as f:
       ML_MODEL = pickle.load(f)
   logger.info('Complete loading model')
```

Bot Technique:
Use ML models to identify anomalous activity or malicious contracts.

Time series analysis, anomaly detection, opcode clustering and analysis.

Relevant bots:

Smart Price Change Detector
https://github.com/0xidase/Smart-Price-Changes-Agent

Time Series Analyzer
https://github.com/forta-network/starter-kits/tree/main/time-series-analyzer-template

Contract Deconstructor
https://github.com/OpenZeppelin/contract-bots-gang/tree/master/contract-deconstruct

Malicious Smart Contract ML Detector
https://github.com/forta-network/starter-kits/tree/main/malicious-smart-contract-ml-py

Section 4

Challenges.

Future areas to research.

- Trusted private scan pools (private bots)
- Pre-submission TX scanning
- On-chain alerts

Known challenges: Atomic attacks, private transactions, monitoring secrecy, response latency.

To learn more or to get involved please visit forta.org.

Thank you!

Jonathan Alexander
CTO OpenZeppelin
jonathan@openzeppelin.com
@jalex206