Detekcia nepovoleného prístupu kustomizáciou nízko interaktívnych honeypotov

Jakub Perdek
Slovenská technická univerzita v Bratislave
Bratislava, Slovensko
perdek.jakub@gmail.com

***Abstract*—Prevencia pri zabezpečení systému obvykle nemusí byť dostatočná. Z toho dôvodu je potrebné detegovať aktivity spojené hlavne s neoprávneným prístupom do systému, súborom alebo do intranetu. Ich skoré odhalenie môže pomôcť prijať vhodné opatrenia a zabrániť reálnemu útoku na systém. Vhodným nástrojom sú práve honeypoty. Tie naviac môžu slúžiť aj na spomalenie neoprávnených aktivít alebo na zmiatnutie útočníka. Často sú ale ľahko rozpoznateľné od reálnych aktív, a preto je ich kustomizácia nevyhnutná. Lákanie možno realizovať podľa viacerých stratégií, a podľa toho aj prispôsobiť generovanie kustomizovateľných tokenov. Predstavujeme preto automatické riešenie pre tvorbu nízko interaktívnych honeypotov založené na čo najväčšej infiltrácii logiky pre sledovanie rozličných foriem manipulovania s nimi v rámci konkrétnej biznis logiky. Snahou je poskytnúť informácie pri záujeme o konkrétny obsah. Analyzované sú preto rôzne spôsoby aplikovania uvedených mechanizmov v rámci webových dokumentov. Možnosti pre zamedzenie ich zneužitia a obmedzenie manipulácie s nimi by v rámci tohto druhu honeypotov mali byť ľahšie dosiahnuteľné a prispôsobené pre konkrétny prípad použitia.**

***Index Terms*—nízko interaktívne honeypoty, detekcia nepovoleného vstupu, kustomizácia honeypotov, webové honeytokeny**

I. Úvod

Honeypoty sú bezpečnostným zdrojom, ktorý generuje upozornenie pri zachytení nadviazania interakcie s ním [9]. Napríklad v podobe prieskumu, útoku alebo pri kompromitácii. Často lákajú infiltrovaný subjekt, a pri ich dobrom maskovaní a umiestnení môžu pomôcť odhaliť neoprávnený prístup, odhalené prístupové údaje, dokonca spomaliť aktivity útočníka vrátane odhalenie nultého útoku a rovnako aj zistiť niektoré ním aplikované postupy pri útoku. Bežní používatelia by nemali s honeypotom vôbec interagovať, ale ani vedieť o tejto funkcionalite. Prihlásenie sa do takéhoto systému alebo otvorenie a manipulácia s dokumentom, respektíve honey tokenom [6] je preto automaticky podozrivá, a malo by byť pomocou oznámení na ňu upozornené. Honeypoty neriešia špecifický problém, a pokiaľ s nimi nikto neinteraguje ich hodnota je pre organizáciu veľmi nízka. V praxi môžu byť realizované rôznymi spôsobmi v rámci orientácie na špecifický cieľ. Tým môže byť aj detekcia neoprávnených aktivít znalých členov v organizácii vrátane nových typov útokov [12].

V našej práci sme sa zamerali na nízko interaktívne webové honey tokeny odosielajúce informácie pri pokuse získať z nich fiktívne biznis informácie. Zamerali sme sa na možné spôsoby realizácie týchto tokenov s čo najmenšou závislosťou na ich umiestnení a procesoch nevyhnutných pre ich sledovanie, tak aby sme mohli zabezpečiť kustomizáciu bez závislosti na externých faktoroch.

II. Honeypoty pre detekciu neoprávneného prístupu

Pre detekciu neoprávneného prístupu v rámci internej siete organizácie je najvhodnejšie použiť produkčné honeypoty [4]. Svojou nízkou mierou false pozitív dokážu identifikovať väčšinu pokusov o vniknutie do internej siete, kompromitáciu systému alebo jeho časti. Druhou možnosťou sú aj výskumné honeypoty ale vzhľadom na ich typické nasadenie s priamim prístupom do verejnej siete sa zvyšuje ožnosť ich kompromitácie kýmkoľvek. Získané informácie o použitých prihlasovacích údajoch a ďalších aktivitách nie sú preto často relevantné. Zároveň by takého honeypoty mali dosahovať vyššiu mieru interaktínosti.

Existujú rôzne techniky oklamania potencionálneho útočníka. Medzi hlavné z nich patria klamlivá služba (Deception service), emulácia celého operačného systému *(OS emulation)*, emulácia zraniteľnosti *(vulnerability emulation)*, tarpitting sieťového spojenia *(connection tarpitting)*, presmerovanie spojenia *(traffic redirrection)* a digitálna návnada *(digital bait)* [7]. Typickým reprezentantom pre techniku digitálnej

návnady sú honey tokeny. Týmito entitami môže byť všetko čo obsahuje falošné informácie [7] ako napríklad fiktívny dokument alebo odkaz na neexistujúcu službu alebo produkt v konfiguračnom súbore inej služby. Manipulácia s takýmito dokumentmi má za následok vygenerovanie varovných hlásení.

Honeypot nástroje sú dostupné ako komerčné alebo aj ako open source riešenia. Príkladmi voľne šíriteľných honeypotov sú napríklad honey klient Thug a SSH honeypot Cowrie. Nízko interaktívny honeypot Thug sa používa na vyhľadanie škodlivého obsahu vrátane exploidov hlavne v javascriptovských súboroch a ďalších webových dokumentoch [15]. Analýzu je možné vykonať priamo skenovaním webovej lokality alebo lokálnych súborov. Iným honey klientom je YALIH simulujúci správanie klienta na konkrétnej webovej lokalite, ktorý rovnako hľadá vzorky škodlivého kódu [2]. Známy stredne interaktívny honeypot je Cowrie. Vďaka svojej schopnosti simulovať súborový systém umožňuje vyššiu interaktivitu s útočníkom. Tým umožňuje získať podrobnejšie informácie o útočníkových aktivitách [1]. K dispozícii sú preň rôzne pluginy a proxy pre protokoly SSH a Telnet. Ďalšie voľne dostupné riešenia možno nájsť v rámci pôvodných projektoch na stránke honeynet.org [1] alebo ich zoznamy aj s popisom a odkazmi na stránkach smokescreen.io [2] [13] a awesome-honeypots [3].

III. Benefity nízko interaktívnych honeypotov pre detekciu neoprávneného prístupu

Medzi úrovňou interakcie honeypotov je často potrebné robiť trade-off. Napríklad nízko úrovňové honeypoty je jednoduché nainštalovať, ľahko emulujú niektoré služby. Riziko je tak nízke ale rovnako je limitovaná aj poskytnutá informácia o aktivite útočníka [11]. Často nevyžadujú zmeny v existujúcej topológii siete alebo na zariadeniach [10]. Nízko interaktívne honeypoty sú väčšinou produkčné honeypoty určené na ochranu organizácie [11]. Naopak vysoko interaktívne honeypoty je väčšinou náročné nainštalovať, nakonfigurovať a nasadiť. Pri použití dokerizácie je ich tvorba automatizovaná a jednoduchá aj pre nešpecializovaného administrátora. Bezpečnostný expert je potrebný aj v tomto prípade, a to kvôli monitorovaniu a nalýze logov vyžadujúcej znalosti slabín konkrétneho nástroja pri filtrovaní aktivít v systéme [14]. Nízko interaktívne honeypoty poskytujúce informácie s nízkou konkrétnosťou nevyžadujú ich zložitú analýzu. Iba upozorňujú na potencionálne nežiadúce aktivity. Pre overenie týchto aktivít sú už ale experti potrební. Vysosko interaktívne honeypoty rovnako poskytujú skutočné služby namiesto simulácií, a preto predstavujú vysoké bezpečnostné riziko pri zneužití. Riešením v prípade detegovania ich zneužitia pre kontajnerizované aplikácie orchestrované pomocou orchestrátora Kubernetes môže byť ich automatické resetovanie [8]. Celkový prehľad porovnania jednotlivých typov honeypotov je vypracovaný v tabuľke uverejnenej v [5].

IV. Návrh nízko interaktívneho honey tokenu z webových dokumentov

Nízko interaktívne honeypoty pre detekciu neoprávneného prístupu by mali zahŕňať mechanizmy pre generovanie informácie o príslušnej udalosti. Tie by mali byť dostatočne maskované, tak aby sa útočník o svojom odhalení najlepšie ani nedozvedel. V svojej práci sme sa zamerali na webové dokumenty.

A. Mechanizmy pre logovanie aktivity s webovým tokenom

Analyzovali sme rôzne spôsoby vloženia spomenutého mechanizmu do webového dokumentu. Zamerali sme sa na prípady, keď sa manipuluje s webovým dokumentom neposkytovaným priamo v rámci nejakého spusteného servera. Výsledný honeytoken v podobe konkrétneho dokumentu bude poskytovať obmedzené možnosti interakcie s ním, a tým sa zabráni aj jeho zneužitiu. Identifikovali sme nasledovné prípady umožnenie sledovania interakcie s uvedeným dokumentom:

1) Použitie nástroja sledujúceho zmeny v rámci súborového systému, prípadne iného systému v rámci ktorého sa uvedené tokeny manažujú, pre vybrané súbory. Získanou informáciou potom je len informácia o záujme o tieto súbory. Detekovanie takejto zmeny môže byť aj jednoduchšie detegovať.
2) Vloženie, respektíve nahradenie alebo upravenie existujúceho iframe elementu novým. Infiltrovanej osobe sa tým priamo poskytne hľadaný obsah aj s odoslaním informácií pri dopytoch po zdrojových súboroch nového obsahu. Zároveň možno odpozorovať o aký obsah má osoba záujem a príslušne tento obsah kustomizovať. Aj napriek tomu, že sa iframy už neodporúča používať stále sa vyskytujú

[1] https://www.honeynet.org/projects/

[2] https://www.smokescreen.io/practical-honeypots-a-list-of-open-source-deception-tools-that-detect-threats-for-free/

[3] https://github.com/paralax/awesome-honeypots

hlavne pri službách poskytujúcich letáky v internetových obchodoch.

3) Vytvorenie a odoslanie informujúcej správy, respektíve logu na server. Správu by malo byť potrebné maskovať. Oproti predchádajúcemu spôsobu môže byť takáto správa odchytená a jej odoslanie na server útočníkom znemožnené, keďže sama o sebe spravidla neposkytuje pre útočníka relevantný obsah. Mala by preto byť prepojená s vyžiadaním konkrétnych kľúčov pre sprístupnenie tohto obsahu.

Obmedzenie na menované spôsoby je hlavne z dôvodu ochrán pred internetovými hrozbami,v rámci ktorých nie je povolené zapisovať na disk pomocou prehliadaču štandardným spôsobom. Rovnako ukladanie je obmedzené na použitie cookies, ktoré si používateľ spravidla môže prezerať. Pri aplikácii preto zostáva informovať s použitím naviazania na zdroje, ktoré sú ku dokumentu priradené.

B. Nástroje pre maskovanie logiky honeypotu a ich význam pre kustomizáciu

Pri tvorbe funkcionality umožňujúcej detekovať manipuláciu s týmito tokenmi a ich ďalšiu kustomizáciu sme použili nástroje pre:

1) Vytvorenie kópie danej webovej lokality, prípadne iba jej časti.
2) Manipulovanie so štruktúrou webového dokumentu a automatickú zmenu informácií v rámci nej.
3) Minifikáciu a zamlženie súborov pre kaskádové štýly, skripty pre javascript a hlavne HTML dokumenty.
4) Využitie proxy, ktoré umožňuje získať a pozmeniť časť pravého obsahu pre nalákanie útočníka.
5) Zahashovanie a zneprístupnenie obsahu až do momentu priameho vykonania skriptov na tránke. Inak by útočník mohol získať želaný obsah iba otvorením konkrétneho súboru.

Vytvorenie kópie webovej lokality sme realizovali pomocou balíčkov prístupných v jazyku python. Zistili sme ale ich obmedzenú funkčnosť pri reálnom použití. V prípade balíčku pywebcopy [4] to bolo časté zamrznutie skriptu kvôli zlúčeniu vlákien. Použili sme preto najnovšiu verziu priamo z githubu [5] a zablokovali použitie multivláknového spracovania ale vyskytol sa nový problém s absolútnymi adresami skopírovaného dokumentu. Kvôli nim sa napríklad nezobrazia pôvodné obrázky na stránke. Ďalším nástrojom bolo vloženie podpory pre HTTrack [6], ktorý je efektívnejší a nemal komplikácie ako predchádzajúci balík. Jeho použitie je ale závislé na operačnom systéme a v súboroch necháva informácie o použití tohto nástroja.

Príklad použitia nástroja HTTrack pre získanie súborov do hĺbky 1, bez externých závislostí, a ich uloženie do priečinku ./downloads:

```
"C:\\Program Files\\WinHTTrack\\
   httrack.exe" https://www.trony.it/
   online/store-locator -v -r1 -\%e0 -
   O ./download
```

Pokiaľ je možné tak manipulujeme so štruktúru webového dokumentu dynamicky s použitím knižnice Beautifulsoup. Typickou použiadavkou je upravenie konkrétnych iframe elementov a nastavenie ich atribútov. Niekedy je potrebné funkcionalitu využiť pre prepojenie aj upravených alebo vytvorených skriptov alebo kaskádových štýlov s webovým dokumentom. Pokiaľ načítanie tejto štruktúry nie je možné malo by byť zabezpečené pridanie uvedených častí v textovej podobe.

Do riešenia sme pridali niekoľko nástrojov pre zamlženie a minfikáciu súborov. Rôzne nástroje poskytovali funkcionalitu pre rozdielne typy súborov. Pre minifikáciu HTML sme použili pythonovskú knižnicu htmlmin [7] s nastavením pre vymazanie prázdneho miesta a odstránenie komentárov. Pre minimalizáciu javascriptu sme použili Closure Compiler [8] od Googlu. V rámci nastavení sme použili compilation-level na SIMPLE_OPTIMIZATIONS, kvôli tomu, že pri pokročilejšom nastavení dochádzalo k odstránenie všetkého kódu pokiaľ nebol priamo exportovaný. Redundantné funkcie určené na zneprehľadnenie kódu by tak boli odstránené. Následne sme pridali parameter pre spracovanie aj javascriptovských súborov v striktnom móde. Minimalizáciu kaskádových štýlov sme zabezpečili pomocou externého nástroja Yui Compressor [9] volaného podobne z príkazového riadku. Jednotlivé nástroje sme aplikovali na príslušné súbory v konkrétnej vybranej stromovej štruktúre. V praxi takýmito súbormi sú súbory klonovanej webvej lokality so zapracovanou sledovacou logikou.

Ďalšou voliteľnú časť tvorí použitie proxy pre poskytnutie časti relevantného obsahu s dodatočnou možnosťou

[4]https://pypi.org/project/pywebcopy/

[5]https://github.com/rajatomar788/pywebcopy

[6]https://www.httrack.com

[7]https://pypi.org/project/htmlmin

[8]https://github.com/google/closure-compiler

[9]https://github.com/yui/yuicompressor

úpravy. Funkcionalitu sme sa rozhodli pridať kvôli uľahčeniu procesu kopírovania časti namiesto celej webovej lokality, keďže v praxi táto činnosť vyžadovala kopírovanie obsahu aj z externých stránok. Reštriktívny je CORS hlavne pri zabezpečenejších aplikáciám. Namiesto proxy je preto výhodnejšie poskytovať celý fiktívny obsah pre honeypot priamo. Pri využití proxy možno reálne dáta z existujúcich služieb upravovať za behu, ale rastie riziko odhalenia takéhoto honeypotu.

Podstatnými sú nástroje pre maskovanie obsahu. Dáta je možné uložiť do viacerých častí, ktoré sa v priebehu vykonania zložia. To núti používateľa aby súbor reálne otvoril v prehliadači, nechal vykonať kód a tým zapríčinil aj vykonanie skrytej sledovacej logiky. Opäť by sprístupnenie obsahu malo spočívať na komunikácii so serverom, inak sledovacia logika neoznámi manipuláciu s tokenom. Rovnako je možné použiť base64 hash, ktorým ukryjeme obsah a hlavne sledovaciu logiku pred vyhľadaním zvonka. Alternatívou je prevod do hexadecimálneho formátu alebo použitie knižnice pre obalenie dát. Existujú rôzne formáty, používaným môže byť gzip.

V. Automatizácia kustomizácie honeytokenu

Kustomizácia je podstatná pre zníženie pravdepodobnosti detekcie honeypotu. V praxi dochádza k fingerprintingu [3], kedy útočník na základe určitých charakteristických znakov odhalí honeypot. Príkladom môžu byť rôzne preklepy alebo nedostatočné schopnosti simulácie reálnych nástrojov. Príkladom môže byť odoslanie chybovej správy so status kódom 200 namiesto príslušného chybového kódu. Imitácia webového servera v tomto prípade zlyhala. Pri overovaní pravosti sa útočník sústredí hlavne na tieto usvedčujúce charakteristické črty. Automatizovanie a zavádzanie náhodnosti pri kustomizácii by malo útočníkom sťažiť schopnosť takejto detekcie.

V rámci automatizácie a kustomizácie honey tokenu riešime dve stratégie. V rámci prvej používateľ špecifikuje počet výrazných honeytokenov a doménu z ktorej majú byť vytvorené. V rámci druhej stratégie automatizácia a kustomizácia nebude zvýrazňovať určité charakteristiky cieľového súboru a jeho obsahu ale vytvorí dostatočný počet takýchto podobných tokenov.

V rámci samotnej automatizácie sa vykonajú kroky v nasledovnej postupnosti:

1) Klonovanie webových dokumentov z danej domény. Zváži sa tu aj klonovanie nepovolených stránok v rámci súboru robots.txt a hĺbka tohto klonovania.
2) Pre každý generovaný prvok sa rovnako vykoná nasledovný postup. Vloží sa element so sledovacím iframe, pri ktorom server loguje požiadavky pre dopytovanie sa po jeho obsahu alebo sa nahradí existujúci iframe aj spolu s presunutím poskytovania tohto obsahu na server. V prípade nepoužitia iframu sa vygeneruje a vloží kód odosielajúci informácie pri vyžiadaní konkrétneho obsahu.
3) Klonovanie webovej lokality na adrese, na ktorú smeruje iframe a tvorba API pre spracovanie funkcionality poskytovanej konkrétnym iframom. Obsah v rámci biznis domény uložený v rámci konkrétnych súborov by mal byť ďalej kustomizovaný. V prípade nepoužitia iframu sa vygeneruje kostra servera poskytujúceho kľúče pre sprístupnenie obsahu z klienta a odmaskovanie správy pre jej zalogovanie.
4) Ďalšie hashovanie a zneprístupňovanie obsahu v rámci súborov. Zabezpečenie funkčnosti pri vykreslení stránky a aplikovaní kľúčov zo serveru.
5) Prípadné pridanie redundantného kódu alebo spájanie a vykonanie zamlženej verzie.
6) Rovnako pre každý generovaný honeytoken prebehne minifikácia a prípadné zamlženie skriptov, html súborov a asociovaných kaskádových štýlov.

VI. Návrh a implementácia nástrojov pre utajenie obsahu

Ukrytie obsahu konkrétnych správ ale aj celých skriptov môže byť v rámci statických súborov nevyhnutné. Zároveň je potrebné aplikovať takýto postup na každú zahrnutú biznis informáciu, ktorá by neskôr mala byť sprístupnená a prezentovaná v rámci konkrétneho honeytokenu. Sprístupnia sa potom už iba konkrétne fiktívne biznis informácie. Útočník by si konkrétnych logov a ich obsahu potom nemal všimnúť. Rovnako by nemal ani neblokovať požiadavky smerujúce na server kvôli jeho potencionálnemu záujmu o biznis obsah. Uvedená funkcionalita by mala zabezpečiť rôzne druhy informácií odosielané z rôznych častí webového dokuemntu a zvýšiť tak možnosti informovania o činnosti útočníka.

A. Utajenie obsahu kompresiou a kódovaním

Pre budúci generátor honey tokenov sme navrhli funkcionalitu, ktorá bude obsah niekoľko ráz maskovať s použitím náhodne vybraných metód uplatnených aj niekoľko krát. Výsledkom bude maskovaný obsah a kľúčom bude postupnosť identifikátorov metód, aby následne mohli byť použité pre navrátenie textu do čitateľnej podoby. Funkcionalitu sme zostrojili tvorbou

triedy podporujúcej dve základné metódy pre maskovanie a odmaskovanie. Tie konkrétne implementácie dediace od nej prekryli vlastnými využívajúcimi svoju funkcionalitu a umožnili tak náhradu za túto základnú triedu, čo dopomohlo k náhodnému výberu konkrétnej metódy. V základne triede sme implementovali nástroje pre náhodný výber metódy, získanie metódy podľa identifikátora, funkcionalitu umožňujúcu maskovať a rovnako odmaskovať konkrétny obsah.

Pri implementácii konkrétnych maskovacích metód sme využili knižnice tretích strán. Okrem základnej funkcionality umožňujúcej vytvoriť z textu hash ako napríklad base64, base32, base16 alebo base85 (knižnica base64 [10]) sme implementovali aj podporu pre kompresné metódy ako je gzip (knižnica gzip [11]), deflate (knižnica zlib [12]) alebo brotli (knižnica brotli [13]). Vstupom ako aj výstupom každej z týchto metód je textový reťazec. Problémom kompresných metód je vrátenie výslednej hodnoty v binárnej podobe. Bolo preto zaručiť konverziu do textovej podoby. To ale v rámci kódovania utf-8 nebolo kvôli rôznym nevyhovujúcim znakom možné. Pri vynechaní chýb nebolo spätné odmaskovanie úplne úspešné a uplatnenie viacerých metód ani možné. Rovnako pretypovanie spôsobovalo chyby pri integrácii spomenutých kompresných metód. Nakoniec sa nám problém podrilo vyriešiť nástrojom pre quotovanie url (knižnica urlib [14]), ktorý to dokázal urobiť aj z binárnej podoby (metóda pre prevod z binárnej podoby na text má podobu urllib.parse.quote_from_bytes(compressed_bytes)
a metóda pre spätný prevod má podobu urllib.parse.unquote_to_bytes(encoded_text)).
Ako ďalšie metódy pre maskovanie textu sme použili funkcionalitu pre quotovanie url (urllib.parse.quote_plus(pure_text)) a jeho spätné dekódovanie (urllib.parse.unquote_plus(encoded_text)).

Nakoniec sme implementovali spomenuté metódy pre maskovanie a odmaskovanie. Metóda pre maskovanie získa na vstupe zvolený text, pole maskovacích metód a počet uplatnených cyklov. Následne náhodne vyberie z tohto poľa maskovaciu metódu a uplatní ju na zvolený text. Zaznamená sa aj druh tejto metódy podľa identifikátora tejto metódy. Postup sa opakuje zvolený počet krát vždy s predtým maskovaným textom. Výsledkom je maskovaný text a identifikátor všetkých použitých maskovacích metód ako kľúč. Ten sa môže umiestniť na serveri a zabezpečiť tak utajenie obsahu, alebo zaručiť odmaskovanie už na klientovi s nevyhnutnou potrebou maskovania tohto kľúču. V rámci uvedenej funkcionality je predpoklad odmaskovania na klientovi dôležitý, inak by postačovala aj iná forma šifrovania, pri ktorej by server bol schopný sprístupniť konkrétne správy, respektíve informácie.

Metóda na odmaskovanie na vstupe požaduje maskovaný text a identifikátor všetkých použitých metód pre maskovanie. V rámci tejto metódy sa vykoná reverzná aplikácia uvedených metód v opačnom poradí ako boli použité pri maskovaní. Výsledkom by mal byť odmaskovaný text v podobe v akej bol pred jeho utajením.

B. Zamlženie kódu jeho rozdelením a vykonaním v rámci druhého kódu

V praxi okrem štandardnej minifikácie kódu zahŕňajúcej náhradu premenných a odstránenia prázdneho miesta môže byť užitočné aj rozdelenie kódu na viaceré časti. Ak sa pred rozdelením aplikuje maskovanie obsahu z predchádzajúcej časti dostávame tak dômyselne utajený obsah. Týmto obsahom by mala byť najčastejšie správa o postupe útočníka ale môže ním byť aj zdrojový kód samotný. V prípade takéhoto spracovania zdrojového kódu samotného ako reťazca je potrebné vykonanie metódy eval. V rámci nej sa zdrojový kód zapísaný ako textový reťazec vykoná. Použitie tejto metódy je stále veľkým bezpečnostným riskom, a preto by sa malo takému použitiu vyhnúť. Ideálne kód predĺžiť rozdelením jednotlivých príkazov, lepším prepletením kódu s bizis logikou alebo pridaním nepotrebných a redundantných častí.

V rámci našej implementácie sme umožnili rozdeliť textový reťazec na viaceré časti s použitím premenných a polí. Do polí sú tieto časti pridávané priamo alebo vo forme premenných, ku ktorým sú predtým priradené. Následne je predtým rozdelený reťazec znovu zložený. Jednotlivé príkazy sú generované zaradom a ukladané do poľa. Náhodne sa pritom aplikujú implementované metódy pre generovanie príkazov ako napríklad tvorba premennej vo forme textového poľa alebo textového reťazca a zlúčenie premenných. Dodatočne je možné pridávať medzi tieto príkazy aj iné redundantné príkazy alebo príkazy prislúchajúce ku konkrétnej biznis logike. Funkcionalitu sme testovali s aplikovaním maskovania obsahu a testovali s vykonaním javascriptu pomocou knižnice js2py [15]. Výstupné texty boli zhodné. Identi-

[10]https://docs.python.org/3/library/base64.html
[11]https://docs.python.org/3/library/gzip.html
[12]https://docs.python.org/3/library/zlib.html
[13]https://pypi.org/project/Brotli
[14]https://docs.python.org/3/library/urllib.html
[15]https://pypi.org/project/Js2Py/

fikovali sme aj problém s použitím deklarácií s použitím let, pri ktorých bolo potrebné konkrétny kód obaliť do funkcie inak mohlo dochádzať ku kolízii premenných. Použitie var namiesto let by problém vyriešilo, ale mohlo by spôsobiť aj problémy a malo by byť používané len výnimočne pre globálne deklarácie.

VII. Vloženie detekčnej logiky

Detekčná logika je hlavným obsahom honeytokenu. Do dokumentu musí byť vhodným spôsobom pridaná a následne je potrebné zabezpečiť zneprístupnenie informácií a zamedziť jej odhalenie.

A. Vloženie detekčného skriptu

Podstatným pre využitie a prepojenie predtým realizovaných metód pre klonovanie, minimalizáciu a utajovanie obsahu je samotné vloženie správy alebo skriptu. To musí byť maskované vzhľadom na požiadavku utajene informovať o konkrétnom obsahu. Celý proces realizujeme v niekoľkých krokoch, pričom využívame náhodné generátory pre vygenerovanie rôznych parametrov určujúci finálny výsledok:

1) Príprava skriptu pre odosielanie hlásení na server. Dodatočne je možné pripraviť aj zoznam správ, ktoré sa budú posielať.
2) Príprava webových služieb pre manažovanie logov a samotnej kostry serveru.
3) Klonovanie príslušnej časti webovej lokality.
4) Načítanie skriptu pre detegovanie podozrivej aktivity, prípadne aj samostatných správ.
5) Zamaskovanie buď celého skriptu, pričom v neskorších fázach bude potrebné použiť funkciu eval pre jeho vykonanie po odmaskovaní. Dôležité je maskovanie jednotlivých správ samostatne kvôli ich utajeniu. Tie budú následne počas vykonávania odosielané.
6) Vloženie metód pre dešifrovanie konkrétneho obsahu s kódom metódy, ale nie samotných správ s dôrazom na maskovanie identifikátora použitých metód. Implementovanie tejto funkcionality priamo na serveri z dôvodu zistenia obsahu logov a ich následného spracovania.
7) Rozdelenie obsahu do niekoľkých riadkov kódu, prípadne metód. Vhodné je aj zahrnúť tento obsah do kódu pre biznis logiku alebo pridať redundantný kód.
8) V prípade maskovania celého skriptu je potrebné po domaskovaní zabezpečiť volanie funkcie eval pre jeho vykonanie.
9) Vloženie celého kódu do script elementu a pripojenie ho do honeytokenu, respektíve webového dokumentu.
10) Otestovanie spustením servera a otvorením honeytokenu. Odoslané hlásenia by mali byť zaznamenané.

B. Injekcia iframe elementu s konkrétnym obsahom

Chýbajúci dynamický obsah možno v rámci statického obsahu honeytokenu vložiť ako vnorený dokument pomocou iframe elementu. V konkrétnom dokumente konkrétnej webovej lokality môžme nahradiť už nejaký existujúci alebo vložiť nový. Pri nahradzovaní existujúceho by sme mali nahradiť aj celú poskytovanú biznis logiku. V niektorých prípadoch pre tieto účely možno použiť proxi server. Druhou možnosťou je propojiť iframe element, ktorý sa nemusí ani zobraziť. V tomto prípade vzrastá potencionálna hrozba detegovateľnosti honeypotu. Poskytnutie reálneho obsahu tak môže byť spoľahlivejšie a zaručí to aj zabránenie blokovaniu požiadaviek na server. Aj v tomto prípade uvádzame postup tvorby takéhoto honeytokenu s dôrazom na náhodné určenie jednotlivých parametrov:

1) Klonovanie príslušnej časti webovej lokality s webovým dokuemntom.
2) Vyhľadanie iframe elementu a zamenenie src atribútu za vlastnú webovú službu. Ak taký eleemnt neexistuje, alebo je komplikované napodobniť inú lokalitu, potom sa vytvorí nový iframe element s nastavenými rozmermi na 0.
3) Pokiaľ bol nahradzovaný obsah iframe elemntu, potom sa zrealizuje klon webovej lokality na ktorú smeroval jeho pôvodný obsah a vloží sa do zdrojov servera alebo sa prispôsobí server tak aby robil proxy medzi pôvodnou lokalitou. V rámci proxy sa niektoré informácie môzu pozmeniť, aby útočník nedostal reálne biznis dáta.
4) Doplnenie potrebného obsahu pre logovanie a ďalších potrebných služieb (napríklad prípadné proxy) na server.
5) Maskovanie zvyšného obsahu vyššie opísanými technikami pre zmiatnutie útočníka.
6) Otestovanie funkcionality spustením servera a otvorením honeytokenu. Správa o prístupe k danému obsahu by mala byť zaznamenaná.

VIII. Automatizované generovanie honey tokenov

Po návrhu kustomizovateľného honey tokenu bolo potrebné realizovať aj ich masové generovanie

zosúladené so zvolenou stratégiou. Vhodné je zabezpečiť aj rozšíriteľnosť pre aplikovanie ľubovoľnej stratégie. Ich tvorbu sme preto rozdelili na niekoľko častí. V prvej fáze sa podľa špecifických kritérií vytvorí JSON konfiguračný súbor s parametrami určujúcimi základné komponenty a ich orientáciu. Túto fázu je možné automatizovať a po vytvorení príslušného súboru manuálne zrevidovať a upraviť výsledné hodnoty. Druhou fázou je samotné vytvorenie konkrétnych inštancií honey tokenov a serverov podľa vytvorenej konfigurácie. Následne rovnako možno manuálne upraviť výsledné časti, tak aby pôsobili čo najdôveryhodnejšie.

Zabezpečenie kustomizácie API pre komunikáciu klienta a serveru v rámci zvolenej stratégie vyžaduje určenie konkrétnych parametrov v konfigurácii pre obidve časti zároveň. Príkladom môže byť konkrétna URL na ktorej bude server získavať konrétne informácie od klienta. Server môže na tejto adrese sprístupňovať zvolený obsah, odmaskovávať skrytý obsah správ a rovnako logovať prichádzajúce správy. Klient musí byť schopný odoslať správu vo vopred navrhnutom formáte na túto URL. Pre každý honey token sa zvolí reprezentujúci odkaz na existujúci webový dokument podľa ktorého sa má vytvoriť. Už v tejto fáze je možné rozhodnúť o minifikácii vybraného obsahu a ďalších doplnkoch a zmenách. Okrem samotného prispôsobenia a falšovania webového dokumentu je pri konkrétnych stratégiách potrebné zvoliť aký skript sa má použiť pri komunikácii so serverom a zároveň pomocou ďalších metód aj zamaskovať. Skript by mal byť písaný v Javascripte a obsahovať pre framework rozpoznateľné pomenovania dôležitých častí, ktoré v rámci konkretneho použitia budú za behu nahradené za hodnoty z konfiguračného súboru. Príkladom takejto časti môže byť už spomínaná URL adresa, ale aj reťazec identifikujúci aplikované maskovacie metódy. Podobným spôsobom sa vytvorí aj funkcionalita samotného serveru. Server môže byť vytvorený v ľubovoľnom programovacom jazyku podporujúcom nástroje pre komunikáciu s týmito dokumentami. Viacero honey tokenov by malo byť schopné komunikácie s jedným serverom. V každom z dokumentov najvyššej úrovne v konfiguračnom JSON súbore sa špecifikujú honey tokeny a práve jeden server s príslušnými rozhraniami pre ne. Vzhľadom na potrebu odovzdania informácií výhradne sieťovou komunikáciu je prítomnosť servera nevyhnutná.

Nami vytvorené riešenie podporuje automatické generovanie častí NodeJS servera. V jednoduchom prototype sa skopíruje nevyhnutná funkcionalita a postupne sa pridávajú ďalšie časti, ktoré sa podľa potreby upravujú podľa hodnôt v konfiguračnom súbore. Samotné metódy pre spracovanie príchodzích dopytov by mali byť rovnako napísané v oddelenom súbore s rozpoznateľnými hodnotami pre používaný rámec. Kód z týchto súborov rámec podľa konfigurácie zahrnie na príslušné miesto. Pri realizácii konkrétnej stratégie je tak potrebné napísať niekoľko takýchto súborov s príslušnou funkcionalitou.

Po vygenerovaní príslušných častí je potrebné ešte doinštalovať potrebné balíčky pre spustenie servera a prípadne vytvorené tokeny vhodne umiestniť. Samotné umiestnenie tokenov a servera je možné nastaviť aj v konfiguračnom súbore. V prípade NodeJS servera je pre inštaláciu možné použiť príkaz

```
npm install
```

a pre jeho spustenie

```
npm start
```

.

Vhodná konfigurácia môže pomôcť hlavne pri tvorbe výrazných honey tokenov lákajúcich svojím vzhľadom. Môže to byť napríklad dodatočné generovanie odkazov na konkrétny token alebo napísanie názvu súboru veľkými písmenami. Prevažná väčšina ostatných súborov by potom už nemala mať podobné prvky a pôsobiť tak pre útočníka nezaujímavo. Iná stratégia založená na podobnosti väčšiny zo súborov je obvykle priamo podporená v tvorbe konfiguračného súboru, keďže existujúce dokumenty obvykle možno rýchlo získať priamo z webovej domény, a zároveň na nich netreba uplatniť kustomizáciu vo vyššej miere. Obsah týchto dokumentov by ale mal byť fiktívny, aby dokumenty nemohli byť ďalej zneužité.

V rámci testovania automatickej tvorby honey tokenov sme napísali krátke konfiguračné súbory pre obidve stratégie. Generátor tak vytvoril 3 tokeny, pričom sa pre každý uplatnil osobitný postup jeho tvorby. Postup je opísaný v predchádzajúcich kapitolách. Zároveň sa pre zadefinovanú skupinu vytvoril server s možnosťou prjímať a spracovať dopyty od týchto tokenov. Vytvorenému serveru sme doinštalovali potrebné balíčky a spustili ho. Následne sme otvárali jednotlive honey tokeny v prehliadači. V konzole serveru následne po každom otvorení takéhoto súboru pribudol jeden záznam. Po otvorení súborov bol obsah skriptov roztrieštený v kóde a na obsah boli aplikované niektoré maskovacie metódy. Tokeny vyzerali rovnako ako pôvodné webové dokumenty.

IX. Zhrnutie a budúca práca

Kustomizácia je pri honeypotoch veľmi dôležitá hlavne kvôli zmenšeniu miery ich detekovateľnosti. Zautomatizovanie procesu ich tvorby spolu so zavedením náhodného výberu spomedzi možných vlastností pomáha urýchliť ich vývoj a znížiť aj úsilie potrebné pre spomenutú kustomizáciu. Navrhli a vytvorili sme preto program umožňujúci nielen vytvoriť a do istej miery aj prispôsobiť konkrétne honey tokeny, ale uplatniť celý proces v rámci konkrétnej stratégie vábenia útočníka. Rozmiestnenie ako aj nasadenie týchto honey tokenov realizujeme niekoľými spôsobmi. Pri niektorých sa predpokladá vytvorenie veľkého množstva podobných inštancií, pri inej len nízky počet tých výrazných. Okrem samotných webových dokumentov sa vygeneruje jeden alebo viac serverov a služby zabezpečujúce logovanie správ z honey tokenov. Riešenie sme obohatili o techniky ukrytia obsahu, logovania správ a stiahnutia predlôh tokenov z konkrétnej webovej lokality pre ich ďalšiu kustomizáciu. Implementácia bola realizovaná podľa požiadaviek zabezpečujúcich detekovanie aktivity záujmu o konkrétne informácie špecifického webového dokumentu v prípade ich sprístupnenia respektíve odtajnenia.

Celkovo sme vytvorili dva zakladné typy honey tokenov. Jeden využívaja iframe elementy a druhý obsahuje funkcionalitu umožňujúcu odosielať správy. Funkcionalita fungujúca ako proxy a pozmeňujúca prebratý obsah nie niekedy funkčná a použiteľná z dôvodu blokovania CORS politikou. Kustomizácia je preto pracnejšia. Ďalšou zistenou nevýhodou je, že iframe elementy sú zastaralé, a často aj podozrivé pokiaľ priamo neprispievajú svojím obsahom na stránke hlavne ak majú nastavený rozmer na nulové hodnoty. Stále sa však v mnohých aplikáciách používajú na sprístupnenie vloženého obsahu akým sú napríklad elektronické letáky pri eshopoch. Zabezpečili sme preto klonovanie upravených informácií, ktoré budú dynamicky pomocou iframe elementu do dokumentu pri jeho otvorení vložené. Druhý typ používa skript pre odosielanie správ. Ten vyžaduje dômyselnejšie utajenie obsahu správy a zamedzenie jej blokovania pri možnom odchytení. Riešenie sme preto obohatili nástrojmi pre minifikáciu webových dokumentov a ich častí, rezdelením kódu na menšie časti a prepletením s biznis logikou, ale aj zapracovaním postupného generovania obsahu až po otvorení dokumentu v prehliadači.

Tvorbu honey tokenov sme parametrizovali a umožnili ich realizáciu podľa konfiguračného súboru. Tieto súbory obsahujú aj parametre jednotlivých webových serverov. Ich tvorba môže byť tiež zautomatizovaná získavaním odkazov na konkrétne časti webovej lokality akými môžu byť konkrétne produkty spolu s následnou kustomizáciou. Hlavným cieľom je doladenie niektorých kustomizačných parametrov pri uplatnení navrhnutej stratégie lákania útočníka v rámci jeho potencionálneho záujmu o konkrétne webové dokumenty v rámci určitej biznis domény. Samotná realizácia dvoch základných stratégií potvrdila možnosť rýchleho vytvorenia vysokého počtu funkčných honey tokenov prispôsobených nielen svojím obsahom ale ladiaca aj ako celok imitujúci potencionálny cieľ v rámci špecifickej domény. Celý proces aj v závislosti od domény vyžaduje pre čo najvyššie zvýšenie dôveryhodnosti celého riešenia vhodnú modifikáciu klonovaných informácií z danej domény ako aj dodatočnú kustomizáciu vygenerovaného riešenia. Kľúčovým je hlavne idntifikovanie útočníkovho záujmu o konkrétne informácie z domény a ich následné zahrnutie do jedného z honey tokenov.

V ďalšej práci by sme chceli pridať viac parametrov pre konfiguráciu webových honey tokenov. Rovnako by sme chceli vhodným spôsobom chceli zabezpečiť čo najednoduchšie použitie proxi v rámci iframe elementov bez zásahov do pôvodnej biznis funkcionality.

References

[1] W. Cabral, C. Valli, L. Sikos, and S. Wakeling. Review and Analysis of Cowrie Artefacts and Their Potential to be Used Deceptively. In *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 166–171, Las Vegas, NV, USA, Dec. 2019. IEEE.

[2] M. Mansoori, I. Welch, and Q. Fu. YALIH, Yet Another Low Interaction Honeyclient. *New Zealand*, 149:9, 2014.

[3] M. Mohammed and H.-u. Rehman. *Honeypots and Routers: Collecting Internet Attacks*. Auerbach Publications, 0 edition, Dec. 2015.

[4] C. Moore and A. Al-Nemrat. An Analysis of Honeypot Programs and the Attack Data Collected. In H. Jahankhani, A. Carlile, B. Akhgar, A. Taal, A. G. Hessami, and A. Hosseinian-Far, editors, *Global Security, Safety and Sustainability: Tomorrow's Challenges of Cyber Security*, volume 534, pages 228–238. Springer International Publishing, Cham, 2015. Series Title: Communications in Computer and Information Science.

[5] B. Nagpal, N. Singh, N. Chauhan, and P. Sharma. CATCH: Comparison and analysis of tools covering honeypots. In *2015 International Conference on Advances in Computer Engineering and Applications*, pages 783–786, Ghaziabad, India, Mar. 2015. IEEE.

[6] C. K. Ng, L. Pan, and Y. Xiang. *Honeypot Frameworks and Their Applications: A New Framework*. SpringerBriefs on Cyber Security Systems and Networks. Springer Singapore, Singapore, 2018.

[7] M. T. Qassrawi and Z. Hongli. Deception Methodology in Virtual Honeypots. In *2010 Second International Conference on Networks Security, Wireless Communications and Trusted Computing*, pages 462–467, Wuhan, China, 2010. IEEE.

[8] D. Reti and N. Becker. Escape the Fake: Introducing Simulated Container-Escapes for Honeypots. *arXiv:2104.03651 [cs]*, Apr. 2021. arXiv: 2104.03651.

[9] C. Sanders. *Intrusion detection honeypots: detection through deception*. 2020. OCLC: 1293961192.

[10] C. Scott. Designing and Implementing a Honeypot for a SCADA Network. page 39, 2014.

[11] L. Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[12] L. Spitzner. Honeypots: catching the insider threat. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pages 170–179, Las Vegas, Nevada, USA, 2003. IEEE.

[13] S. TEAM. Open Source Honeypots That Detect Threats For Free.

[14] M. Valicek, G. Schramm, M. Pirker, and S. Schrittwieser. Creation and Integration of Remote High Interaction Honeypots. In *2017 International Conference on Software Security and Assurance (ICSSA)*, pages 50–55, Altoona, PA, July 2017. IEEE.

[15] N. F. Zulkurnain, A. F. Rebitanim, and N. A. Malik. Analysis of THUG: A Low-Interaction Client Honeypot to Identify Malicious Websites and Malwares. In *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*, pages 135–140, Kuala Lumpur, Sept. 2018. IEEE.